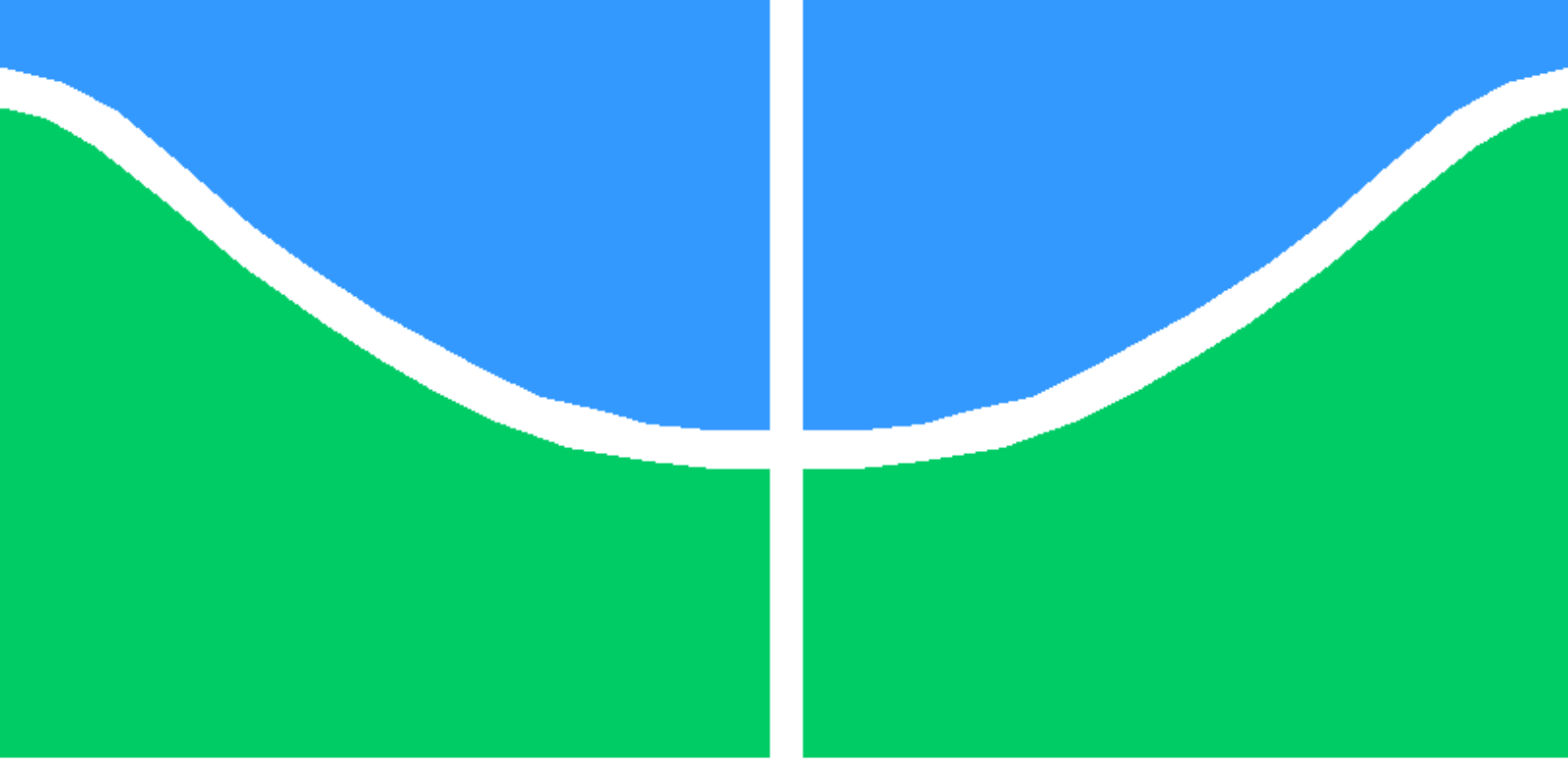

MONITORAMENTO DE BEBÊS EM TEMPO REAL VIA ARQUITETURA DE INTERNET DAS COISAS

LEONARDO CORRÊA FOSSI

LUCAS GARCIA REIS FERREIRA

PROJETO FINAL DE GRADUAÇÃO EM ENGENHARIA DE REDES DE COMUNICAÇÃO

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA ENGENHARIA DE REDES DE COMUNICAÇÃO

MONITORAMENTO DE BEBÊS EM TEMPO REAL VIA ARQUITETURA DE INTERNET DAS COISAS

LEONARDO CORRÊA FOSSI

LUCAS GARCIA REIS FERREIRA

Orientador: PROF. DR. MARCELO MENEZES DE CARVALHO, ENE/UNB

PROJETO FINAL DE GRADUAÇÃO EM ENGENHARIA DE REDES DE COMUNICAÇÃO

PUBLICAÇÃO -
BRASÍLIA-DF, 25 DE MAIO DE 2021.

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

MONITORAMENTO DE BEBÊS EM TEMPO REAL VIA ARQUITETURA DE INTERNET DAS COISAS

LEONARDO CORRÊA FOSSI

LUCAS GARCIA REIS FERREIRA

PROJETO FINAL DE GRADUAÇÃO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM ENGENHARIA DE REDES DE COMUNICAÇÃO.

APROVADA POR:

Prof. Dr. Marcelo Menezes de Carvalho, ENE/UnB
Orientador

Prof.ª Dra. Flavia Maria Guerra de S. A. Oliveira, ENE/UnB
Examinador interno

Prof. Dr. Paulo Roberto de Lira Gondim, ENE/UnB
Examinador interno

BRASÍLIA, 25 DE MAIO DE 2021.

Agradecimentos

Agradecemos, primeiramente, por todo apoio e auxílio prestados pelas nossas famílias. Agradecemos, com todo respeito, aos professores do Departamento de Engenharia Elétrica. Agradecemos, em particular, ao nosso professor e orientador Dr. Marcelo Menezes de Carvalho. Obrigado pela paciência e por sua orientação. Agradecemos também aos nossos amigos e colegas de curso, que nos acompanharam nesses cinco anos de UnB.

Resumo

Este trabalho apresenta um sistema não intrusivo de baixo custo para monitoramento de bebês em tempo real via arquitetura baseada no paradigma de Internet das Coisas. O sistema visa o monitoramento multidimensional de grandezas físicas que auxiliam a detecção de eventos e condições ambientais considerados alarmantes e que demandam atenção ao bebê. Os sinais são coletados via duas placas acopladas a diversos sensores capazes de fornecer informações sobre o ambiente e a criança. O kit XDK da Bosch é responsável pelos dados referentes ao ambiente — temperatura, umidade e luminosidade — enquanto que o microcontrolador ESP32 é responsável pela coleta de informações referentes ao comportamento da criança, envolvendo som e movimento. Todos os dados coletados são enviados para um servidor na nuvem, onde são analisados e com isso, eventos são detectados e alertas enviados. O envio de alertas é feito por meio do aplicativo de mensagens Telegram. Desta forma, os responsáveis recebem notificações em seus dispositivos celulares informando-os sobre eventos detectados pelo sistema. O desempenho do sistema é avaliado a partir da execução e apresentação de um experimentos baseados no monitoramento de um bebê por várias horas. Os resultados indicam que o sistema proposto tem desempenho satisfatório para as condições avaliadas, alcançando acurácia maior que 58,3% na detecção de todos os eventos contemplados.

Abstract

This paper presents a non-intrusive low-cost system used for monitoring babies in real time via an architecture based on the Internet of Things paradigm. The system aims to monitor physical quantities in multiple dimensions in order to detect events and environmental conditions that are considered alarming and demand attention to the baby. The signals are collect by two micro-controllers attached to several sensors capable of gathering information about the child and the environment. The XDK development kit from Bosch is responsible for the environmental data — temperature, humidity and lighting — whilst the ESP32 micro-controller is responsible for the information regarding the child's behavior, including sound and movement. All the data collected is then sent to a cloud server where they are analyzed so that events can be detected and alerts can be sent. Alert notification is done via the Telegram messaging app. Thus, the people responsible for the child receive notifications on their cellphones warning them about events detected by the system. The system's performance and efficiency are measured from the execution and presentation of an experiment based on the monitoring of a baby for several hours. The results indicate that the proposed system shows satisfactory perfomance for the given conditions, reaching accuracy higher than 58.3% for the detection of all contemplated events.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS FONTE

Capítulo 1

Introdução

1.1 Contextualização

Desde a década de 1970, quando as redes de computadores ainda estavam sendo implementadas, a Internet evoluiu muito. Com a difusão do uso da eletrônica digital e da produção em massa de circuitos lógicos digitais, computadores foram se tornando cada vez mais acessíveis até o ponto em que quase todos possuem um celular e um computador em casa. O sistema ALOHA, que começou como uma necessidade, abriu caminho para tecnologias incríveis dos dias de hoje que talvez em um ou dois anos já se tornem obsoletas, como as redes 4G, o WiFi, a transmissão de dados via fibra óptica, etc. Como já era de se esperar, paralelo ao desenvolvimento tecnológico, a sociedade também evolui. Hoje há tantas preocupações que o tempo se tornou um recurso escasso. A exigência por produtividade cresce em uma curva tão íngreme que o auxílio da tecnologia torna-se uma necessidade.

Neste contexto, surge uma demanda por informações rápidas e precisas a fim de facilitar as tarefas do cotidiano. Emerge então um novo conceito: Internet das Coisas — IoT (do inglês, *Internet of Things*), que representa uma solução em potencial para melhoria da vida das pessoas. A IoT é a rede de dispositivos físicos, veículos, eletrodomésticos, sensores e software, que permite que essas "coisas" se conectem e trabalhem juntas com troca de dados em tempo real. O termo "Coisa" em "Internet das Coisas" pode ser, por exemplo, uma pessoa com monitor de respiração, um animal com um chip de localização, um veículo que possui sensores para alertar o motorista quando o mesmo está apresentando características de sonolência, ou sensores integrados com o berço de um bebê recém nascido que monitoram a criança e alertam os pais caso a criança necessite de atenção. Todas essas "Coisas" podem ter um endereço atribuído, ganhando assim a capacidade de transferir dados através de uma rede. Como resultado, está se tornando cada vez mais fácil integrar o mundo físico aos sistemas baseados em computador, o que resulta em melhorias, aumento de eficiência, benefícios econômicos e redução do esforço humano.

1.2 Tema

Dentro deste cenário de evolução tecnológica e vida trabalhosa, os pais têm cada vez menos tempo para dedicarem aos seus filhos. Apesar de alguns aspectos da criação serem impossíveis de serem melhorados pela tecnologia, outros podem. Entre eles, existe o monitoramento de crianças na fase em que são mais vulneráveis e demandam mais cuidado: os primeiros meses de vida. Segundo o Centro de Controle e Prevenção de Doenças dos Estados Unidos [8], a morte súbita infantil inesperada, em inglês — *Sudden Unexpected Infant Deaths (SUID)*, é um termo usado para descrever a morte súbita e inesperada de um bebê com menos de um ano de idade em que a causa não era óbvia antes da investigação. Essas mortes geralmente acontecem durante o sono ou no ambiente de sono do bebê. Mortes infantis inesperadas incluem a síndrome da morte súbita do lactente (SMSL), em inglês — *Sudden Infant Death Syndrome (SIDS)*, sufocamento acidental e outras mortes de causas desconhecidas. Em 2019, nos Estados Unidos, houve cerca de 1.250 mortes devido a SMSL, cerca de 1.180 mortes devido a causas desconhecidas e cerca de 960 mortes devido a sufocamento acidental [8]. A síndrome da morte súbita do lactente é a principal causa relatada de morte pós-neonatal no mundo desenvolvido, segundo Edwin Mitchell [9], e tem aumentado de forma constante desde a década de 1980, da mesma maneira que o sufocamento acidental de crianças. Carlin e Moon [10] acreditam que a SMSL ocorre em bebês com vulnerabilidade biológica e citam os perigos que o ambiente inseguro pode oferecer. Para eles, configura ambiente inseguro dormir de bruços ou de lado, roupa de cama macia e compartilhamento de cama. Concluem afirmando que um ambiente seguro é fundamental para a redução de risco de SMSL.

Segundo Nunes e Pinho [11], o risco de SMSL é maior quando os lactentes dormem na posição prona (de bruços), ou seja, a posição em que o bebê é colocado para dormir aparece como um dos fatores de risco de SMSL. Sendo assim, realizou-se uma campanha em que foi estimulada a mudança da posição de dormir de prona para supina e, imediatamente, foi observada uma redução da mortalidade pós-natal em até 50% nos países que adotaram a posição supina para dormir. Ainda segundo o artigo citado, apesar de algumas crianças já nascerem com risco maior de SMSL e do mecanismo da morte ainda não ser totalmente conhecido, acredita-se que, ao fornecer um ambiente de sono mais propício, elimina-se os fatores "gatilho" das criança vulneráveis, reduzindo, consequentemente, a mortalidade. Posto isto, torna-se evidente que o uso de ferramentas que auxiliam os pais a monitorar seus filhos e o ambiente, em tempo real, é de extrema importância. Para mitigar os riscos de SUID, acredita-se que a intervenção célere dos pais ou dos responsáveis, ao serem alertados de que algo está errado, pode ser o diferencial. Segundo Messaoud e Tadj [12], o choro é a linguagem que os bebês utilizam para comunicarem suas necessidades aos pais, que devem satisfazê-los oportunamente, com o objetivo de acalmá-los ou auxiliá-los em situações adversas, como sede, fome ou desconforto. Neste trabalho, o objetivo é: alertar os responsáveis pela criança, caso o bebê demonstre sinais de incômodo ou caso as condições ambientais não es-

tejam ideias para o bebê, por meio de um sistema automático e não intrusivo de análise de sinais de áudio, movimento, luminosidade, umidade e temperatura.

1.3 Objetivo

Tendo em vista a necessidade de agilidade no atendimento às crianças que dão sinais de incômodo (para assim reduzir os riscos de SMSL), o presente projeto tem como objetivo alertar os responsáveis pelo bebê assim que um evento for detectado. Os usuários são alertados independentemente de sua distância em relação à criança por meio de notificações recebidas em seus *smartphones*. Quando longe, podem avisar alguém que esteja perto para prestar socorro, caso o responsável imediato não responda. Sem o uso de câmeras nem de monitores, os pais podem saber se a criança está agitada, muito quieta, se está emitindo algum som e podem também acompanhar as condições ambientais (umidade, luminosidade e temperatura). As babás eletrônicas tradicionais, usadas comumente no monitoramento de crianças, utilizam câmeras e monitores com limites de alcance e de usuários, além de exigirem o monitoramento ativo dos usuários, isto é, os responsáveis devem periodicamente conferir a transmissão de vídeo para verificarem se a criança está bem. Este projeto propõe um sistema de alertas que pode ser compartilhado, não só pelos pais, mas por qualquer pessoa que queira ajudar a cuidar do bebê, sem a necessidade de compartilhamento de equipamento. O usuário final recebe em seu aparelho celular notificações de detecções de:

- Gemidos;
- Gritos;
- Choros;
- Ausência de movimentos detectados por um período prolongado;
- Agitação incomum (movimentação contínua por um período prolongado);
- Temperatura do ambiente abaixo ou acima da faixa ideal para o bebê;
- Umidade do ar abaixo do nível ideal para o bebê;
- Alta luminosidade no ambiente;

O objetivo do sistema é facilitar a tarefa de monitoração da criança por meio de um sistema IoT não intrusivo que mantenha o seu berço sob observação contínua, sem precisar da visualização ativa e contínua dos responsáveis. Ao deixarem o bebê desacompanhado em seu berço, o

sistema passa a monitorar características do ambiente, como temperatura, umidade e luminosidade, bem como o comportamento da criança, isto é, se ela está emitindo algum som, se apresenta movimentação incomum ou se não apresenta movimentação alguma. Com isso, alerta-se os pais sempre que o bebê precisar de atenção.

1.4 Revisão Bibliográfica

Muito já tem sido feito em termos de sistemas de monitoramento de bebês baseados em IoT. O funcionamento dos sistemas varia bastante no que diz respeito ao monitoramento de bebês, mas poucos apresentam interesse na utilização de recursos em nuvem, como no caso deste projeto. No trabalho desenvolvido por Liu et al. [13], é apresentado uma solução baseada em monitoramento por vídeo que utiliza uma técnica chamada de *Eulerian Magnification*, capaz de amplificar as diferenças entre quadros de vídeo, possibilitando a percepção das mesmas. A câmera é posicionada de forma a cobrir todo o berço do bebê e é utilizada para acompanhar os movimentos torácicos da criança. Quando as diferenças entre quadros param de ser detectadas, sabe-se que a respiração do bebê está alterada, então os pais são alertados. Este trabalho tem alguns pontos fracos, sendo o maior deles o fato de que a detecção não é feita em tempo real. De acordo com os autores, o sistema não está pronto para utilização devido a isto, mas é a prioridade deles em trabalhos futuros. Outro ponto relevante é o fato de que o monitoramento contínuo por vídeo pode representar alto consumo de energia e a transferência de dados de vídeo em alta resolução pode gerar um grande sobrecarga na rede.

Um trabalho semelhante é apresentado por Dubey e Damke [14], em que também utiliza-se processamento de imagem. Neste trabalho, a posição do bebê no berço é reconhecida a partir do processamento da imagem capturada e o choro é detectado utilizando um sensor de ruído ambiental similar ao utilizado neste projeto, contudo o sensor utilizado no projeto supracitado faz uso de saída digital, que contém menos informação que o sensor de saída analógica. Logo, análises dos sinais não poderão ser realizadas. Quando choro é detectado ou se o posicionamento do bebê está muito próximo dos limites do berço, uma foto do berço é enviada por e-mail aos pais, junto com uma mensagem de alerta. Um dos pontos fracos deste trabalho é a detecção de choro, que é baseada apenas em um único limiar de detecção definido no módulo de ruído ambiental. Se um único pico é detectado no sensor, o alerta e a foto são enviados. Além disso, a utilização contínua da câmera representa os mesmos custos citados anteriormente em termos de uso de energia e sobrecarga na rede.

Hussain et al. [15] utilizam em seu trabalho uma técnica conhecida como Gráfico de Controle (do inglês *Control Charts*) que determina estatisticamente uma faixa denominada limite de controle delimitada por uma linha superior (limite superior de controle) e uma linha inferior (limite inferior de controle), além de uma linha média. O objetivo é verificar, por meio do gráfico, se o

processo está sob controle, isto é, determinar se um processo é ou não estável ou tem desempenho previsível. Assim sendo, os autores definiram limiares de intensidade e diversidade dos movimentos da criança baseando-se em um conjunto de dados próprios. O mesmo método é utilizado para monitoramento da respiração do bebê. A detecção é bastante complexa e confiável, mas um sistema eficiente de alertas não é detalhado no trabalho. Além disso, os mesmos problemas de uso de energia e uso de banda estão presentes aqui.

No trabalho desenvolvido por Zakaria et al. [16], foi implementado um sistema capaz de monitorar a temperatura da criança por meio de um termômetro alojado na meia do bebê. A tendência de variação de temperatura pode ser acompanhada por um aplicativo móvel e os alertas são enviados por um *Bot* do Twitter. Além dos alertas, um *buzzer* dispara uma notificação sonora para os adultos próximos ao bebê. Este trabalho faz o armazenamento e o processamento dos dados na nuvem, assim como o projeto apresentado no presente documento. A solução é intrusiva e exige que o bebê vista a meia o tempo todo, o que pode ser um problema, principalmente em climas tropicais como o do Brasil. Além disso, esta solução atualmente monitora apenas a temperatura e os alarmes sonoros podem assustar o bebê.

Mandke et al. [17] desenvolveram um sistema de sensores incorporados em uma arquitetura IoT capaz de monitorar frequência cardíaca, temperatura, umidade relativa, som e movimento. As informações são apresentadas em uma página web em forma de tabela. Além de ser um sistema bastante intrusivo, que envolve a utilização de um equipamento de monitoração de batimentos cardíacos que pressiona o dedo do bebê e, na verdade, é projetado para um adulto. O monitoramento de som e movimento é feito por meio de sensores digitais que indicam se há ou não há som e/ou movimento, sem informação precisa sobre o bebê.

Goyal e Kumar [18] construíram um berço que pode balançar automaticamente quando detecta choro e para de balançar quando o bebê está quieto. Possui também um sistema de alerta baseado em alarme de áudio que avisa o usuário quando o colchão está molhado, indicado que deve-se trocar a fralda do bebê e o colchão. O outro alarme é ativado quando o bebê não para de chorar por certo tempo, assim os pais podem verificar o que ocorreu com a criança. No entanto, esse sistema funciona apenas em cenários em que os pais estão próximos, ou seja, o pai e a mãe que trabalham fora não podem monitorar seu filho enquanto estão fora de casa e quando estão em casa podem não escutar o alarme. Outro problema é que o uso de alarme sonoro pode assustar o bebê.

Ishak et al. [19] trabalharam em um sistema de monitoramento para incubadora neonatal. Nele, dois sensores são utilizados: um sensor de batimento cardíaco e um sensor de umidade. Através do Arduino, os dados são enviados para um computador, os quais podem ser analisados pela equipe médica da Unidade de Tratamento Intensivo. Este sistema é projetado para enviar alarmes sempre que as leituras atingirem um nível perigoso, e os testes foram realizados em bebês de zero a doze meses de idade. No entanto, os dados registrados não são transferidos para a Internet. O processamento é feito em um computador conectado diretamente aos sensores, sem conexão à rede. Esta abordagem pode ser melhorada com a adição de um módulo Wi-Fi para

enviar dados pela Internet ao servidor em nuvem e, assim, monitorar os bebês em qualquer lugar e a qualquer hora.

Já no trabalho de Lobo et al. [20], temos um sistema bastante completo capaz de detectar diversos eventos diferentes de forma não intrusiva. Nele, é empregado um sistema de detecção de umidade do colchão, movimento do bebê, choro, temperatura e umidade do ambiente, além de ser capaz até de propor um possível motivo para o choro baseado no sinal de áudio e de vídeo. Além das detecções, em sua maioria frutos do uso de algoritmos baseados em aprendizado de máquina (mais especificamente, *deep learning*), o sistema também aciona um servo motor que balança o berço em caso de choro, bem como um ventilador no caso da temperatura do ambiente estar muito alta. Consideramos este trabalho o estado da arte.

Nossa proposta dispensa o uso de câmeras e do custoso processamento de imagens, tornando assim o projeto mais barato em termos de hardware e mais eficiente no uso de energia e largura de banda. De forma não intrusiva, o sistema é capaz de detectar gemidos, gritos e choros baseando-se apenas nos sinais de áudio e movimento. Também é possível detectar comportamento incomum, isto é, agitação ou ausência de movimento por longos períodos de tempo. Além das detecções de eventos que indicam a necessidade de atenção ao bebê, também monitora-se a temperatura, a umidade do ar e a luminosidade para que assim, os responsáveis possam ser avisados quando alguma ação deve ser tomada para tornar o ambiente mais confortável para a criança. Outro aspecto importante é que várias pessoas são alertadas simultaneamente, ou seja, caso ocorra algum incidente, com o cuidador da criança, os pais podem ser avisados e acionar alguém que esteja próximo como um vizinho ou até mesmo o Serviço de Atendimento Móvel de Urgência.

1.5 Metodologia

A fim de alcançar os objetivos mencionados na sub-seção anterior, é necessário que sejam definidas as tecnologias a serem utilizadas. Tanto os aspectos de hardware, como os serviços utilizados e o software desenvolvido devem ser analisados ao definirmos a arquitetura do projeto.

Em termos de hardware, definiu-se que seria usada o kit, Bosch XDK, mencionado na Seção 3.1.1, como dispositivo responsável pela coleta de dados dos sensores responsáveis por prover informações do ambiente, e então enviá-los para um *gateway*, que enfileira e encaminha os pacotes para o processamento em nuvem, onde seriam finalmente analisados para a geração de alertas. A Bosch XDK trata-se de um kit de desenvolvimento programável em C ou Mita (uma linguagem própria) que possui diversos sensores "embutidos". O equipamento conta com um acelerômetro, um giroscópio, um magnetômetro, um sensor óptico e um sensor ambiental que mede temperatura, pressão e umidade relativa. O dispositivo também possui um módulo WiFi que é utilizado para comunicação com o *gateway*.

Devido às limitações da plataforma Bosch XDK, que impedem a utilização do sensor acústico

simultaneamente aos demais sensores, foi utilizada também o microcontrolador ESP32 [21]. Com o funcionamento muito similar a um Arduino, esta placa é controlada pelo módulo ESP8266EX, possuindo então conectividade WiFi nativa. Conectados ao microcontrolador ESP32, estão os sensores acústico e o sensor de movimento infravermelho conhecido como PIR (*Passive Infrared*). A partir destes sensores, é possível determinar os níveis de ruído no ambiente, bem como se há ou não movimentação na área observada (dentro do carrinho ou berço). Da mesma forma que a plataforma Bosch XDK, os dados coletados pelo microcontrolador ESP32 são enviados via WiFi para o *gateway*.

O *gateway* utilizado não se tratou de um equipamento de hardware isolado, mas sim de pacotes de software (conhecidos como "contêineres") que podem ser executados em um computador conectado à mesma rede local em que se encontram os dois dispositivos utilizados no sensoriamento (kit Bosch XDK e o microcontrolador ESP32). Seu objetivo é centralizar e organizar o envio de dados recebidos das plataformas de monitoração para processamento em nuvem. Fazendo uso da ferramenta conhecida como RabbitMQ [22], o *gateway* armazena mensagens — contendo dados dos sensores — em filas de onde são retiradas e enviadas à nuvem assim que estiverem disponíveis.

O processamento dos dados consiste numa aplicação RESTful [23] desenvolvida sobre a plataforma Flask [24] responsável pela recepção dos dados provenientes do *gateway*, recebidos via requisições HTTP. As informações oriundas do *gateway* são armazenados em um banco de dados MySql, que fica na nuvem, para posteriormente serem processados. Neste servidor é implementada a estratégia de detecção dos eventos de interesse e também o mecanismo de envio de mensagens de alerta para os dispositivo dos pais. Os alertas são enviados por um *chatbot* do aplicativo de mensagens Telegram [25].

Por fim, o sistema é testado no monitoramento em tempo real de um bebê com 2 meses de idade. O equipamento é posicionado no carrinho onde o bebê repousa de forma que não o deixe desconfortável e nem traga riscos à sua saúde. No período de observações a criança observada ainda estava dormindo no quarto dos pais, ao lado da cama, razão pela qual os testes foram realizados no carrinho adaptado para moisés (cama semelhante a uma cestinha, mini-berço ou berço portátil especificamente projetada para bebês desde o nascimento até cerca de quatro meses). Então, o bebê é observado enquanto o sistema faz as detecções. Desta forma, é possível que sejam feitos registros e comparações entre eventos observados presencialmente e alertas gerados pelo sistema. Com um tempo de resposta de menos de dois segundos, foi possível atingir uma acurácia de mais de 60% na detecção de gemidos, gritos, choros/gemidos prolongados, movimentação incomum e ambiente impróprio (temperatura ou umidade inadequada, bem como se há luz no quarto).

1.6 Estrutura do Documento

Este documento está estruturado em 6 capítulos, sendo este o primeiro. No Capítulo 1, é apresentado o contexto em que vivemos, descrevendo o desenvolvimento da Internet e da IoT. Em seguida, comenta-se o tema que nos inspirou a trabalhar neste projeto: a segurança do bebê e o perigo que a síndrome da morte súbita do lactente (SMSL) representa. Posteriormente, apresenta-se o objetivo deste projeto. Depois, descrevemos de forma resumida o funcionamento do sistema e como os resultados são obtidos. Por fim, apresenta-se uma revisão bibliográfica que resume vários trabalhos realizados nesta área.

No Capítulo 2, apresentamos alguns conceitos básicos que foram utilizados como fundamentação teórica e são chave no entendimento do trabalho. No Capítulo 3, fazemos um detalhamento profundo dos materiais utilizados no projeto. Todos os componentes de hardware, software e ferramentas de terceiros utilizados são especificados e seus funcionamentos detalhados. Já no quarto capítulo, apresenta-se a infraestrutura do sistema, isto é, como os materiais estão sendo empregados para que o monitoramento funcione. O funcionamento de todos os componentes do sistema é detalhado neste capítulo, desde a captura e envio dos dados até a detecção de eventos e envio de alertas.

No quinto capítulo, apresentamos e discutimos os resultados da montagem da infraestrutura física e dos testes realizados. Uma comparação é feita entre os eventos observados e os eventos notificados pelo sistema a fim de quantizar a eficiência e a confiabilidade do sistema. Por último no Capítulo 6, apresentamos as conclusões tiradas deste projeto, comentamos os desafios encontrados durante o desenvolvimento do trabalho que impediram que algumas funcionalidades e recursos fossem utilizados. Ainda neste capítulo são discutidos os trabalhos futuros a serem realizados sobre este projeto, contendo funcionalidades que gostaríamos de implementar posteriormente.

Capítulo 2

Fundamentação Teórica

Para tornar possível a compreensão do projeto, é necessário que alguns conceitos-chave sejam fixados. Esse capítulo apresenta os principais conceitos nos quais esse projeto foi fundamentado. Assim, apresentamos as tecnologias mais estudadas, que formam a base teórica para a compreensão do desenvolvimento do sistema de Internet das Coisas.

2.1 Internet das Coisas

O termo "Internet das Coisas" (do inglês, *Internet-of-Things*), foi cunhado por Kevin Ashton ao utilizá-lo como nome de uma apresentação que fez na *Procter & Gamble*(P&G) em 1999. Em seu discurso, ele ressalta o fato de a Internet ser extremamente dependente de informações geradas por pessoas. "Pessoas têm tempo, atenção e precisão limitados, o que as tornam ruins em capturar dados sobre o mundo real." [26]

"A tecnologia IoT pode ser simplesmente explicada como uma conexão entre humanos - computadores - coisas. Todos os equipamentos que usamos no dia-a-dia podem ser controlados e monitorados usando IoT" [27]. Se tivermos computadores que sabem tudo sobre as "coisas" do mundo real, podemos usá-los para coletarmos informações sem a ajuda de pessoas e assim monitorar e controlar qualquer coisa, autonomamente.

Tudo isso é possível graças ao emprego de sensores. Estes equipamentos são de extrema importância para a implementação da IoT pois é por eles que "o ambiente é digitalizado". Os sensores coletam informações do ambiente que permitem que "as coisas" percebam os seus arredores de uma forma ou outra, assim como humanos fazem por meio dos sentidos. A partir das informações fornecidas por sensores, os computadores podem analisar, interpretar e tomar decisões, tudo sem depender da intervenção das pessoas.

Segundo Chen, Jia e Li [28], a arquitetura IoT pode ser tipicamente dividida em 3 domínios:

domínio sensorial, domínio de rede e domínio de aplicação. No domínio sensorial estão os sensores e microcontroladores, responsáveis pela coleta de informações "brutas". A Internet se situa no domínio de rede, interfaceando os domínios sensorial e de aplicação. Por último, temos o domínio de aplicação que engloba o local remoto onde os dados são processados e serviços são providos. É neste contexto que surge um conceito importante para este trabalho: o *gateway*.

O *gateway* é uma interface entre o domínio sensorial e o domínio de rede. Ele recebe mensagens dos sensores, as organiza e as encaminha para a rede pública, eventualmente alcançando um servidor na nuvem responsável pelo processamento dos dados e geração de alertas.

2.2 Computação na Nuvem

Computação em Nuvem é um termo muito popular hoje em dia que representa um conceito bastante simples. Computação na Nuvem, ou *Cloud Computing* é a entrega de recursos de TI sob demanda por meio da Internet com definição de preço de pagamento conforme o uso [29]. Basicamente, trata-se da abstração da necessidade de obter e manter uma infraestrutura física por meio de um serviço provido pela Internet. Os provedores de serviços em Nuvem como a *Amazon Web Services* [30], a *Microsoft Azure* [31] e a *Google Cloud Platform* [32] fornecem opções de capacidade computacional, armazenamento e banco de dados sem que o cliente tenha que se preocupar com a infraestrutura física. Os serviços são cobrados de acordo com o uso, e o "tempo em atividade" (do inglês *uptime*) dos recursos é garantido pelo provedor. Existem ainda 3 tipos diferentes de serviços de computação na nuvem:

- **Infraestrutura como Serviço (IaaS):** o provedor fornece toda a infraestrutura para que o cliente faça uso.
- **Plataforma como Serviço (PaaS):** o provedor fornece uma plataforma onde as aplicações podem ser executadas, abstraindo a necessidade do cliente de manter a infraestrutura adjacente (hardware e sistemas operacionais).
- **Software como Serviço (SaaS):** o provedor fornece uma (ou mais) aplicação(ões) específica(s) já em execução. Ele se responsabiliza por toda a manutenção da aplicação e infraestrutura e o cliente se preocupa apenas em como utilizar tal aplicação.

2.3 Sistemas Microcontrolados

Inicialmente, os computadores eram programados para executar determinadas operações por instruções dadas em código de máquina. Este código consiste em uma série de valores binários e a complexidade exigida para a programação de simples operações era extremamente grande.

Conforme a tecnologia evoluiu e as funções dos programas foram ficando mais complexas, surge o Assembly, uma linguagem de programação de baixo nível que utiliza mnemônicos, isto é, abreviações para operações básicas. O Assembler é a entidade responsável pela conversão do código Assembly em código de máquina, que pode então ser interpretada pelo computador. Décadas depois, passam a surgir as linguagens de alto nível, com capacidades cada vez maiores de abstrair código de máquina para um código facilmente compreensível por humanos. É neste contexto que se encontram o Python e o C, linguagens utilizadas neste trabalho. Essas novas linguagens são utilizadas na programação de diversos tipos de computadores, como por exemplo, **microcontroladores**.

Um microcontrolador é um computador contido em um único circuito integrado que contém um processador e memória. O processador é responsável pela execução do programa, ou programas, e a memória é usada para diversas coisas, como armazenar o código da aplicação ou valores das variáveis e o "estado" da execução. Na maioria das vezes, o microcontrolador comporta em seu circuito integrado um componente de memória disponibilizado na forma de **RAM** - *Random Access Memory* - utilizada como memória primária pelos processadores, isto é, ela é utilizada no armazenamento de dados cruciais para o funcionamento de aplicações que estão em execução. Os endereços são acessados de forma "aleatória", o que quer dizer na verdade "não sequencial", tornando a leitura e a escrita muito mais rápidas do que outras tecnologias como em fitas magnéticas. É importante mencionar também que a memória RAM é uma memória volátil, ou seja, os dados são perdidos quando o dispositivo é desligado.

Há ainda o conceito de **sistema embarcado**. De forma simplificada, um sistema embarcado é um sistema microcontrolado dedicado à aplicação para a qual ele foi programado. Ou seja, enquanto o sistema está em funcionamento, ele realizará apenas as funções predefinidas durante o seu desenvolvimento e a alteração, remoção ou adição de novas funções envolve a reprogramação do controlador. Sistemas embarcados são diferentes de computadores pessoais, por exemplo, que são considerados computadores de propósito geral. Neste projeto, dois subsistemas embarcados diferentes são empregados na coleta e envio de dados dos sensores.

Computadores de propósito geral normalmente contam com uma BIOS e um Sistema Operacional, duas camadas de software que auxiliam no gerenciamento de recursos e no interfaceamento entre usuário e máquina. A BIOS (*Basic Input and Output System*) fica armazenada em memória ROM (um tipo de memória de acesso aleatório não volátil) e é responsável pela inicialização do Sistema Operacional. O Sistema Operacional faz o gerenciamento de hardware e software, além de prover serviços a aplicações. Sistemas embarcados, por outro lado, podem fazer uso ou não de um tipo de sistema operacional conhecido como sistema operacional de tempo real (do inglês, *real time operating system* — **RTOS**) [33].

Apesar da maioria dos Sistemas Operacionais aparentarem executar múltiplas tarefas ao mesmo tempo, na realidade, cada núcleo de processamento executa uma única *thread* por vez. O que dá a ilusão de paralelismo é a entidade conhecida como *scheduler*, que faz o chaveamento

de execução entre os programas. O tipo do Sistema Operacional é definido a partir do comportamento do seu *scheduler*: alguns fornecem tempos de execução iguais para todos os programas, outros dão prioridade para o tempo de resposta apresentado ao usuário, etc. No RTOS, o *scheduler* é projetado para prover um padrão de execução previsível (às vezes descrito como determinístico). Em sistemas embarcados é comum que determinadas operações requiram um tempo de resposta bem definido (às vezes o mais próximo de tempo real possível), o que torna a configuração de prioridades para as *threads* um recurso importante. É por meio desta configuração que um RTOS atinge *determinismo* [33].

Em ambos os microcontroladores empregados neste projeto é utilizado o **FreeRTOS**, um dos RTOS mais utilizados do mundo graças a sua portabilidade entre sistemas diferentes e a sua licença gratuita. As principais vantagens de se usar um Sistema Operacional são: a abstração da responsabilidade de operações de acesso ao *hardware*, como gerenciamento de memória e alocação de núcleos para a execução de tarefas; e a portabilidade de aplicações entre sistemas executando o mesmo Sistema Operacional.

2.4 Protocolo HTTP

O HTTP — Protocolo de Transferência de Hipertexto *(do inglês, HyperText Transfer Protocol)*, protocolo da camada de aplicação da Web, é definido no [RFC 1945] e no [RFC 2616]. Em 1999, a *World Wide Web Consortium* e a *Internet Engineering Task Force* coordenaram a publicação de uma série de *Requests for Comments*; mais especificamente o RFC 2616, que definiu o HTTP/1.1. Em 2015, foi divulgado o lançamento do HTTP/2. A atualização fez com que o navegador passasse a ter um tempo de resposta melhor e mais segurança. Ele também melhorou a navegação em *smartphones*.

Segundo Kurose e Ross [34], o HTTP é executado em dois programas: um cliente e outro servidor. Os dois, executados em sistemas finais diferentes, conversam entre si por meio da troca de mensagens HTTP. O HTTP define a estrutura dessas mensagens e o modo como o cliente e o servidor as trocam. O HTTP usa o TCP como seu protocolo de transporte, sendo assim o cliente HTTP primeiro inicia uma conexão TCP com o servidor. Uma vez estabelecida a conexão, os processos do navegador e do servidor acessam o TCP por meio de suas interfaces de *socket*. Quando um usuário requisita dados ao servidor na nuvem, o cliente envia ao servidor mensagens de requisição HTTP. O servidor recebe as requisições e responde com mensagens de resposta HTTP que contêm as informações solicitadas.

As especificações do HTTP [RFC 1945; RFC 2616] incluem as definições dos formatos das mensagens HTTP. Há dois tipos delas: de requisição e de resposta.

EXEMPLO DE MENSAGEM DE REQUISIÇÃO HTTP

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

A primeira linha é denominada linha de requisição; as subsequentes são denominadas linhas de cabeçalho. A linha de requisição tem três campos: o do método, o do URL e o da versão do HTTP. O campo do método pode assumir vários valores diferentes, entre eles GET, POST, HEAD, PUT e DELETE.

EXEMPLO DE MENSAGEM DE RESPOSTA HTTP

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 09 Aug 2011 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html
(Mensagem Principal ...)
```

A mensagem de resposta HTTP tem três seções: uma linha inicial ou linha de estado, seis linhas de cabeçalho e, em seguida, o corpo da entidade, que é a parte principal da mensagem.

O protocolo HTTP define um conjunto de métodos de requisição responsáveis por indicar a ação a ser executada para um dado recurso. São eles:

- *GET:* é usado quando o cliente requisita recurso ao servidor;
- *POST:* usado para envio de dados ou formulários HTML ao servidor;
- *PUT:* cria ou modifica algum recurso do servidor;
- *HEAD:* é semelhante ao GET. Porém, deixa de fora o recurso requisitado. Esse método é usado com frequência para depuração;
- *DELETE:* permite que o cliente, ou uma aplicação, elimine um recurso no servidor Web.

2.5 Aplicação RESTful

Conhecido como *Representation State Transfer* (REST), é um modelo de arquitetura e não uma linguagem de programação, que fornece diretrizes para que os sistemas, hospedados na nuvem, se comuniquem diretamente com o cliente usando os princípios e protocolos existentes da Web, REST representa uma maneira mais simples de trocar dados entre clientes e servidores através do HTTP [35]. Foi proposto por Roy Fielding e inspirou o documento de arquitetura do W3C *Technical Architecture Group*, que é visto por muitos como um modelo de como construir serviços Web. A *Web REST* é o subconjunto da WWW (com base em HTTP) em que os agentes fornecem semântica de interface uniforme — essencialmente criar, recuperar, atualizar e excluir — em vez de interfaces arbitrárias ou específicas do aplicativo [23].

Sendo assim, pode-se afirmar que uma aplicação RESTful é, basicamente, uma aplicação implementada conforme as indicações do trabalho de Fielding [35]. A API REST simplifica o desenvolvimento de aplicações WEB, ao oferecer suporte a métodos HTTP padrão, tratamento de erros e outras convenções RESTful. Também expõe recursos por meio de URLs, conhecidas como terminais de comunicação (*endpoints*, em inglês). Pode-se usar esta plataforma leve fornecida pela API REST para obter os dados do servidor, fazer alterações no banco de dados, inserir alguma informação ou deletar, sempre que necessário, usando a comunicação por HTTP que faz uso dos métodos: GET, POST, PUT e DELETE.

2.6 JavaScript Object Notation

JSON, ou *JavaScript Object Notation*, é um formato legível mínimo para estruturar dados. Ele é usado principalmente para transmitir dados entre um servidor e um aplicativo da web, como uma alternativa ao XML. As duas partes principais que constituem o JSON são as chaves e os valores. Juntos, eles formam um par chave / valor[36].

- **Chave:** uma chave é sempre uma *string* definida entre aspas;
- **Valor:** um valor pode ser uma *string*, um número, uma expressão booleana, um vetor ou um objeto;
- **Par chave / valor:** um par chave-valor segue uma sintaxe específica, com a chave seguida por dois pontos seguida pelo valor. Os pares de chave / valor são separados por vírgulas.

Abaixo segue um exemplo de resposta em formato JSON depois de consultar o servidor Coronavirus COVID19. Foram requisitados os dados sobre Covid 19 na Suíça.

EXEMPLO DE JSON


```
{
"Country": "Switzerland",
"CountryCode": "CH",
"Confirmed": 20505,
"Deaths": 666,
"Recovered": 6415,
"Active": 13424,
"Date": "2020-04-05T00:00:00Z",
"LocationID": "628d4f12-6ebe-4fa9-b046-77ff0b894a4e"
}
```


2.7 Contêineres

Como Rajdeep, Raja e Kakadia[37] explicam, "contêineres" possibilitam o isolamento de recursos de gerenciamento em um ambiente computacional. O termo é derivado do transporte de contêineres, que é um método padrão para armazenar e transportar qualquer tipo de carga. Um contêiner no sistema operacional fornece uma maneira genérica de isolar um processo do resto do sistema.

Segundo a *International Business Machines Corporation - IBM* a compartimentalização de software em contêineres tornou-se uma tendência importante no desenvolvimento de software como alternativa da virtualização. Esta prática envolve encapsular ou empacotar o código do software e todas as suas dependências para que possa ser executado em qualquer outra infraestrutura.

O processo de encapsulamento em contêineres permite criar e implantar aplicativos com mais agilidade e segurança. Com os métodos tradicionais, o código é desenvolvido em um ambiente de computação específico que, quando transferido para um novo computador, geralmente resulta em erros. Por exemplo, quando um desenvolvedor transfere código de um computador para uma máquina virtual ou de um uma máquina executando o sistema operacional Linux para outra executando o Windows. O uso de contêineres elimina esse problema agrupando o código do aplicativo junto com os arquivos de configuração, bibliotecas e dependências necessárias para sua execução. Esse pacote único de *software* ou "contêiner" é abstraído do sistema operacional hospedeiro e, portanto, torna-se portátil e capaz de ser executado em qualquer plataforma, livre de problemas.

O conceito de contêiner e isolamento de processo tem décadas, mas o surgimento do *Docker Engine* [38] de código aberto em 2013, um padrão da indústria para contêineres com ferramentas de desenvolvedor simples, acelerou a adoção dessa tecnologia.

Os contêineres compartilham o núcleo do sistema operacional da máquina hospedeira e não

exigem a sobrecarga de associar um sistema operacional dentro de cada aplicativo. Os contêineres são inerentemente menores em capacidade do que uma máquina virtual e exigem menos tempo de inicialização, permitindo que muito mais contêineres sejam executados na mesma capacidade de computação de uma única máquina virtual. Isso aumenta a eficiência do servidor que, por sua vez, reduz os custos de recursos e licenciamento. Outro aspecto importante é que aplicativos em contêineres possuem mais segurança, pois podem ser executados como processos isolados e podem operar independentemente de outros contêineres. Isso pode evitar que qualquer código malicioso afete outros contêineres ou invada o sistema hospedeiro.

Por fim, pode-se dizer que o processo de encapsulamento em contêineres permite que os aplicativos sejam "escritos uma vez e executados em qualquer lugar". Essa portabilidade é importante em termos de processo de desenvolvimento. Ele também oferece outros benefícios, como isolamento de falhas, facilidade de gerenciamento e segurança [39].

2.8 Troca de Mensagens

"Troca de mensagens" (do inglês *Messaging*) é um conceito que envolve a utilização de um Mediador de Mensagens (do termo em inglês *Message Broker*) para realizar o gerenciamento do envio de informações entre componentes de aplicações distribuídas. Trata-se do enfileiramento e envio síncrono ou assíncrono de mensagens entre aplicações, em que as mensagens ficam armazenadas até que sejam retiradas da fila. Este conceito é de extrema importância para sistemas distribuídos porque garante que a mensagem não seja perdida até que seja recebida. O funcionamento é ilustrado pelo diagrama apresentado na Figura 2.1.

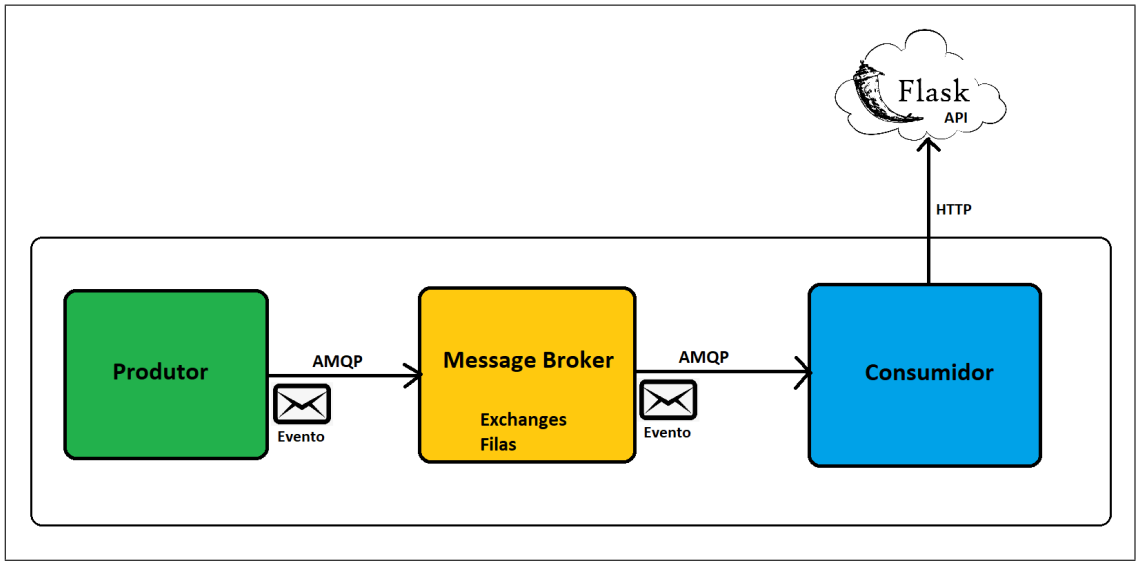

Figura 2.1: Esquemático de funcionamento típico da comunicação por mensagens entre aplicações.

Na Figura 2.1, estão apresentadas todas as entidades envolvidas na comunicação via mensagens. São elas:

- **Produtor:** aplicação que produz e envia as mensagens (também chamadas de *eventos*) ao Mediador de Mensagens.
- **Mediador de Mensagens (*Message Broker*):** servidor responsável pelo gerenciamento e armazenamento das mensagens.
- **Consumidor:** aplicação que observa as filas do Mediador de Mensagens, retira as mensagens prontas e então as processa.

A *Flask API* que aparece na imagem refere-se à API desenvolvida para este projeto que opera na nuvem. Além disso, associado aos eventos aparece o termo "AMQP", que significa *Advanced Message Queueing Protocol*, um protocolo em nível de aplicação utilizado no envio de mensagens/eventos.

Alguns conceitos importantes que ainda não foram detalhados e outros que não constam na Figura 2.1 precisam ser esclarecidos:

- **Fila:** fila gerenciada e armazenada no Mediador de Mensagens. Nela ficam armazenadas as mensagens produzidas pelo Produtor até que o Consumidor as processe.
- **Permutador (do inglês *Exchange*):** uma entidade que atua no Mediador de Mensagens e tem a finalidade de "rotear" as mensagens recebidas do Produtor para a fila ou filas. Dependendo da aplicação, as mensagens podem ser roteadas a uma única fila específica ou a várias (ou ainda a todas as filas).
- **Associação (do inglês *binding*):** ligação estabelecida entre Permutador e fila.
- **Chave de Roteamento (do inglês *routing key*):** atributo associado à mensagem utilizada no roteamento das mensagens realizado pelo Permutador.

É importante para este projeto o entendimento sobre os tipos de Permutador disponíveis para uso no modelo AMQP 0-9-1 [40]:

- **Direto (*Direct*):** a entrega da mensagem é feita de acordo com a passagem de chaves de roteamento (do inglês *routing keys*). Ao fazer uma publicação, além do conteúdo da mensagem, é passado também um atributo *routing key*, que indica ao Permutador para qual fila a mensagem deve ser roteada. Este foi o tipo de Permutador utilizado neste trabalho;
- **Disperso (*Fanout*):** a entrega da mensagem é feita para todas as filas com as quais o Permutador estabeleceu uma associação;

- **Tópico (*Topic*):** a entrega das mensagens é feita a partir de padrões encontrados no atributo *routing key* (chave de roteamento). Quanto utiliza-se este tipo de Permutador, espera-se mensagens com chaves de roteamento compostas por vários termos e o roteamento é feito baseado nisto. Por exemplo, se o padrão é a chaves de roteamento conter uma informação de qual cidade a mensagem é originada e uma informação de aplicação de destino, pode-se configurar uma fila interessada apenas em mensagens provenientes de uma cidade e outra interessada apenas em mensagens destinadas a uma aplicação específica. Se uma mensagem chegar com uma chave de roteamento que indica que é proveniente da cidade e destinada à aplicação em que ambas as filas estão interessadas, a mensagem é repassada para as duas.
- **Cabeçalhos (*Headers*):** entrega as mensagens baseando-se em "cabeçalhos" em vez da chave de roteamento. Mais de um cabeçalho pode ser atribuído a uma mensagem e a uma fila.

Capítulo 3

Materiais e Ferramentas Utilizadas

Na constituição da infraestrutura necessária à implementação do sistema foram utilizados componentes de hardware e ferramentas de software desenvolvidas por terceiros, que encontram-se disponíveis no mercado, além de aplicações desenvolvidas pelos autores do projeto. O kit da Bosch foi a única ferramenta de captura da dados no início da implementação do sistema, entretanto algumas limitações de *hardware* e *software*, dentre entre elas, a impossibilidade de utilização do sensor acústico em paralelo com os outros sensores disponíveis, levaram os autores a optar pela inclusão do microcontrolador ESP32 no projeto, o qual controla a coleta realizada pelos sensores de som e de movimento, descritos nas Subseções 3.1.2.2 e 3.1.2.3, respectivamente. Estes sensores são posicionados ao lado do bebê, enquanto o kit da Bosch captura as informações de ambiente com os sensores de temperatura, umidade e luminosidade. Este capítulo está divido em duas seções, as quais descrevem primeiramente os componentes de *hardware* e posteriormente os componentes de *software* utilizados neste projeto.

3.1 Hardware

Os componentes de *hardware* utilizados são listados a seguir:

- Kit Bosch - XDK

  Módulo de sensores ambientais — BME280

  Sensor de Luz Ambiente — MAX44009

- Microcontrolador ESP32

  Sensor Acústico — Microfone MAX9814

  Sensor de Movimento Infravermelho Passivo — PIR (do inglês, *Passive Infrared*)

Cada um destes componentes terá seu funcionamento geral detalhado nas próximas seções.

3.1.1 Kit Bosch - XDK

A Figura 3.1 mostra o kit da Bosch, responsável pela coleta dos dados de ambiente, que é um dispositivo de sensor programável e plataforma de prototipagem IoT com um kit de desenvolvimento de software — SDK (do inglês, *Software Development Kit*). A plataforma conta com vários sensores:

- Acelerômetro (BMA280)
- Giroscópio (BMG160)
- Magnetômetro (BMM150)
- Luz ambiente (MAX44009)
- Acústico (AKU340)
- Temperatura, Pressão e Umidade (BME280)

Apesar de disponibilizar todos estes sensores, o sensor acústico AKU340 não pode ser utilizado enquanto os demais sensores estão habilitados [41]. Devido a isto, não é possível utilizar o kit XDK para realizar a coleta de todos os sinais que nos interessam.

Figura 3.1: Bosch XDK [1].

O kit da Bosch tem como objetivo conectar o ambiente de forma inteligente, oferecendo um hardware integrado que traz os dados necessários para IoT. De maneira simplificada, a IDE da Bosch (XDK Workbench) e suas bibliotecas permitem que a programação da placa seja acessível e o aprendizado seja rápido[42].

3.1.1.1 Características da Bosch XDK

O hardware utiliza um microcontrolador de 32 bits conhecido como ARM Cortex-M3 e conta com 128 kB de memória RAM e 1 MB de memória Flash, além de já possuir módulos de comunicação integrados - WiFi e Bluetooth. O sistema operacional de código aberto, conhecido como FreeRTOS, é incorporado à arquitetura para prover confiabilidade e segurança às aplicações.

A memória Flash é utilizada somente para o armazenamento do código da aplicação já que ela não é apagada ao realizar-se o *reboot* da placa. Os 128 kB de memória RAM são então utilizados para a execução do programa e devem armazenar a Heap - memória "fixa" utilizada para guardar dados globais - e Stacks - memória dinâmica utilizada por cada tarefa definida na aplicação. Além das estruturas de dados "Heap"e "Stack", a memória RAM é compartilhada também com o núcleo do FreeRTOS. Na Figura 3.2 pode-se observar a distribuição da memória RAM disponível no kit da Bosch.

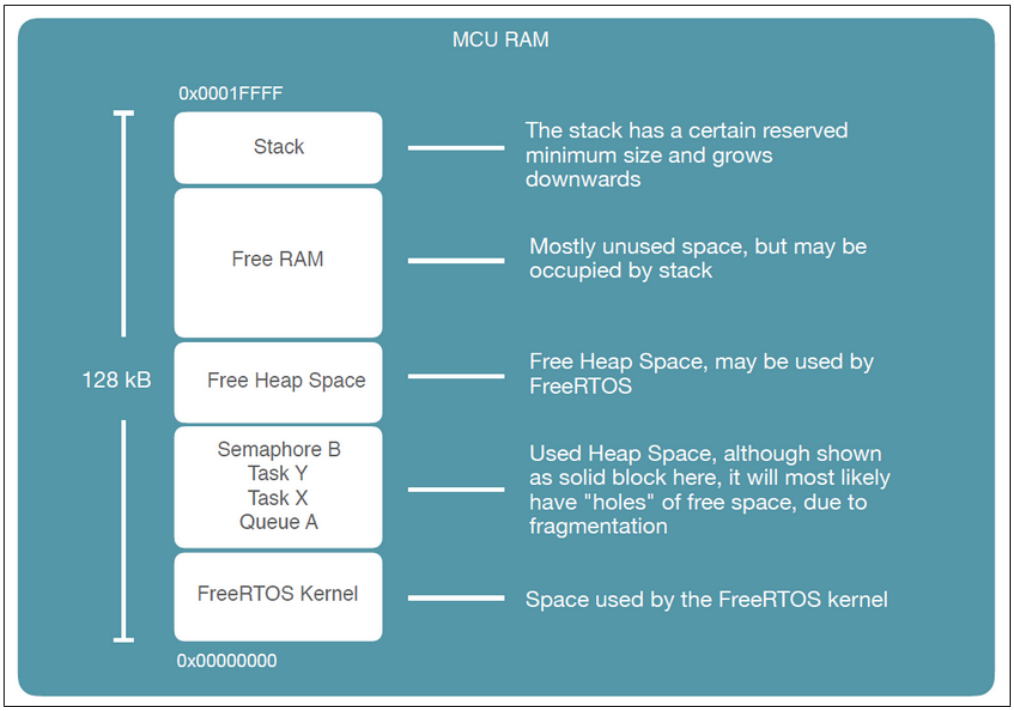

Figura 3.2: Distribuição da memória RAM na Bosch XDK. Pode-se observar como os 128 kB de memória são distribuídos entre: Stack, espaço livre, Heap e núcleo do FreeRTOS [2].

Na prática, a limitação de espaço na memória RAM pode ser um problema para o projeto. Para não sobrecarregarmos a rede e o processamento no gateway e na nuvem, a estratégia pensada envolve o armazenamento de dados periodicamente coletados dos sensores em grandes agrupamentos para então serem enviados. Quanto maior este agrupamento, mais espaço é ocupado na memória RAM. Este foi um dos motivos de não termos utilizado a XDK para envio de dados de áudio, que são coletados em frequência mais alta. Tivemos alguns casos de colisão entre *stack* e *heap*, ocasionando erros na aplicação.

3.1.1.2 Sensores ambientais: Temperatura, Umidade e Pressão - BME280

O kit Bosch XDK contém uma série de sensores embutidos. Um deles, conhecido como um "sensor ambiental", é o BME280, que gera dados de temperatura, pressão e umidade do ambiente. Trata-se de um módulo digital extremamente pequeno e eficiente. As medições de temperatura e umidade podem ser feitas de -20 a 60 ºC e 10 a 90% respectivamente, limitados apenas pelas condições de funcionamento da XDK, enquanto as medições de pressão podem ser feitas entre 300 e 1100hPa [43].

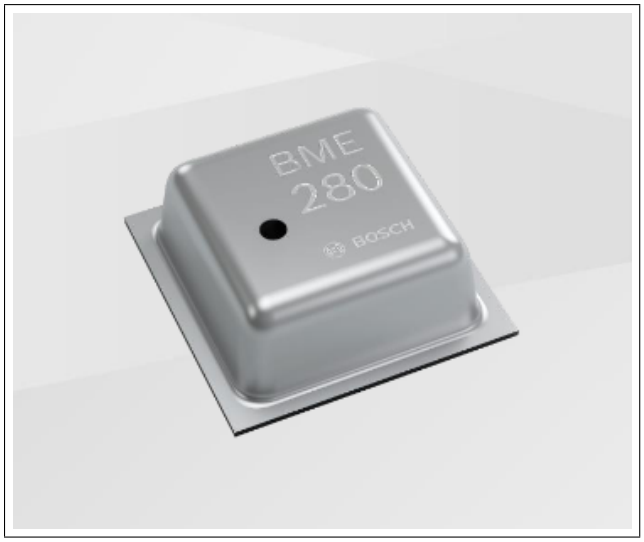

Figura 3.3: Módulo BME280 [3].

Os dados de temperatura são entregues como números inteiros pelo sensor na unidade mºC, representando uma precisão de três casas decimais em termos de graus Celsius. O sensor de umidade também entrega números inteiros ao processador, num formato de proporção (porcentagem), sem casas decimais.

3.1.1.3 Sensor de Luz Ambiente - MAX44009

O módulo MAX44009 também faz parte do kit da Bosch e traz a funcionalidade de monitoramento de luz ambiente. O sensor é conhecido por conseguir operar com correntes baixíssimas de menos de $1\mu A$, o que o torna o sensor de menor consumo de energia do mercado.

A saída do sensor retorna a *iluminância* sobre o equipamento em mililux, que representa o fluxo luminoso (medido em mililúmens) por metro quadrado. O fluxo luminoso pode ser entendido como uma medição de luz visível presente, enquanto a iluminância leva em conta o quanto desse fluxo está incidindo sobre uma área de um metro quadrado. A resposta espectral do fotodiodo usado no sensor é otimizado para imitar o olho humano e conta com proteção contra radiação

infravermelha e ultravioleta. O chip ainda tem um bloco de ganho adaptável que automaticamente seleciona o intervalo de valores de lux correto para otimizar a sensibilidade [44].

A figura 3.4, mostra a abertura de acrílico transparente que o kit contém para que o sensor MAX44009 possa fazer a coleta de luz ambiente. Este sensor é capaz de fazer medições de 22 bits, contemplando os valores entre 0,045 e 188.000 lux [44].

Figura 3.4: Posição do sensor de Luz Ambiente - MAX44009, sob o acrílico no kit da Bosch [1].

Os dados deste sensor são apresentados para o programa como valores inteiros representativos de iluminância em mililux, o que equivale a 3 casas decimais de precisão em termos de lux.

3.1.2 Microcontrolador ESP32

O microcontrolador ESP32, mostrado na Figura 3.5, é um chip com Wi-Fi e Bluetooth de 2,4 GHz, com 30 pinos *GPIO*, sem sensores embutidos, utilizado neste projeto para receber os dados dos sensores de áudio e movimento para enviá-los à nuvem em formato JSON. Este microcontrolador foi escolhido pela facilidade em carregar códigos, por sua unidade de processamento de dois núcleos, baixo consumo de energia, baixo custo, módulo Wi-Fi embutido e 512 KB de memória RAM para instruções e dados.

Figura 3.5: Microcontrolador ESP32 [4].

3.1.2.1 Características do Microcontrolador ESP32 [4]

- Clock de 160 MHz
- Memória RAM estática interna de 520 KB
- Interface Wi-Fi 802.11 b/g/n – 802.11 n (2.4 GHz), até 150 Mbps
- Alimentação VCC de 2,3V a 3,6V CC
- Consumo de corrente máxima com Wi-Fi – 240 mA

3.1.2.2 Sensor Acústico — Microfone MAX9814

A Figura 3.6 apresenta o sensor acústico MAX9418, que é uma placa com um microfone de eletreto capaz de detectar sinais na faixa de frequência entre 20Hz e 20KHz, sendo que o módulo é projetado com um circuito integrado. De forma mais detalhada, o circuito integrado garante ao módulo controle automático de ganho (AGC) para equilibrar o valor lido na entrada, ou seja, quando o som detectado é excessivamente alto, o controle o abaixa o ganho para que não haja uma sobrecarga no amplificador, e em caso contrário, em que a onda sonora está muito fraca, o amplificador aumenta o ganho para que seja realizada a leitura. Além disso, é possível configurar o ganho máximo por decibéis.

Figura 3.6: Microfone MAX9814 [5].

No CI do módulo, ocorrem três amplificações, o pré amplificador de baixo ruído (LNA) tem um ganho fixo de 12dB, enquanto que no segundo amplificador, o ganho VGA se ajusta automaticamente de 0dB a 20dB, de acordo com o limite do AGC. O amplificador de saída oferece ganhos variáveis de 8dB, 18dB e 28dB, de acordo com a configuração do Gain. Na Figura 3.7 pode-se observar diagrama de blocos do CI, retirado do seu respectivo *datasheet* [5].

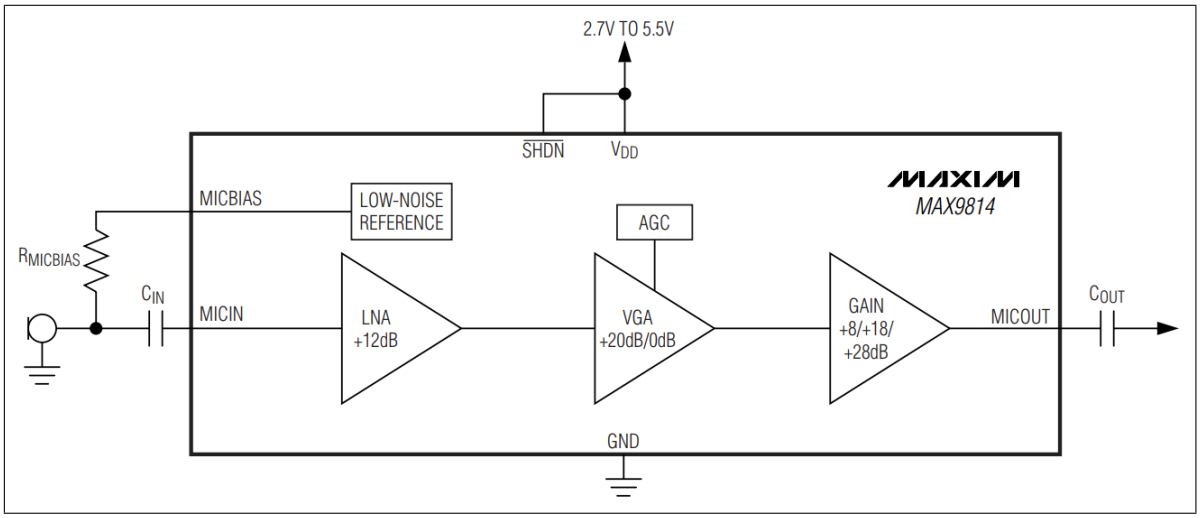

Figura 3.7: Diagrama MAX9814 [5].

3.1.2.3 Sensor de Movimento Infravermelho Passivo — PIR

Um sensor de movimento PIR, mostrado na Figura 3.8, é, basicamente, uma câmera infravermelha que detecta a radiação IR (“radiação de corpo negro”) que é irradiada por objetos que penetram em seu campo de visão. O “P” da sigla PIR significa *passive* (passivo), indicando que o dispositivo não gera nem irradia energia IR em seu processo de detecção, operando exclusivamente por meio da detecção de energia infravermelha emitida pelos objetos. É um sensor de baixo

custo que pode detectar movimentos de seres humanos ou animais. Este sensor tem três pinos: *Vcc, Out e Gnd*, conforme mostrado no diagrama de pinos da Figura 3.9.

Figura 3.8: Sensor de Movimento PIR [6].

O sensor trabalha com dois modos de operação, Repetível(H) e Não Repetível(L). No modo Não Repetível, o pino de saída *Out* fica alto (3,3V) quando algum movimento for detectado dentro do alcance do sensor e permanecerá alto enquanto existir movimento dentro do alcance. Uma vez que não há movimento na área de alcance do sensor, a saída *Out* reduz a zero após um tempo específico que pode ser definido usando o potenciômetro de *Off Time Control*. No modo Repetível, o pino de saída *Out* fica em nível lógico alto (3,3V) quando o movimento é detectado dentro do alcance e fica baixo após um determinado tempo. Neste modo, o pino de saída fica em nível alto, independentemente se há movimento ou não, enquanto o tempo definido não terminar. Os modos de saída do sensor podem ser configurados colocando em curto os pinos à esquerda do módulo, como visto na Figura 3.9 [6].

Neste projeto optou-se pelo modo Não Repetível, pois a emissão de alertas referentes a eventos sonoros (gemidos, gritos e choros) foi concebida de forma que haja adição de informações oriundas do sensor de áudio e do sensor de movimento, além das detecções de agitação incomum e ausência de movimento dependerem totalmente da monitoração da continuidade (ou ausência) dos movimentos. É importante para os algoritmos de detecção apresentados nas Subseções 4.4.2.2 e 4.4.2.3 que as capturas de movimentação sejam feitas de forma contínua, isto é, se o movimento é prolongado, a saída deve se manter em nível lógico alto. Por isso, utiliza-se o modo Não Repetível com o menor tempo configurável em nível alto, que é de três segundos, aproximadamente. Na prática, isto significa que enquanto houver movimento, a saída se mantém em nível alto até que não haja mais movimentação. No momento que o sensor para de detectar movimentação, a

saída se mantém em 3,3V por mais 3 segundos e somente se não houver mais movimentos neste período, a saída retorna ao nível baixo de 0V.

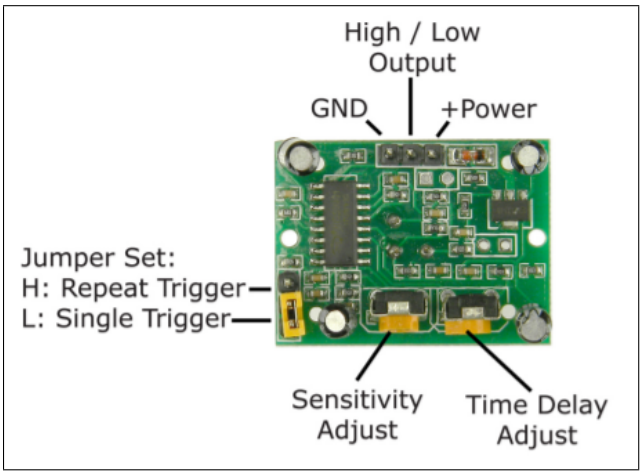

Figura 3.9: Pinagem do Sensor de Movimento PIR [6].

3.2 Ferramentas e Plataformas

Além dos componentes de *Hardware*, alguns componentes de *Software* também foram utilizados neste trabalho. Ambos os códigos utilizados na programação dos controladores foram escritos em C com o auxílio do XDK-Workbench e da Arduino IDE. As aplicações que são executadas no *Gateway* e na nuvem foram escritas na linguagem Python. Foi utilizada uma série de ferramentas de terceiros que encontram-se disponíveis ao público, a fim de facilitar e/ou tornar possível o desenvolvimento do sistema. São elas:

- **RabbitMQ [22]:** *software* gerenciador de mensagens trocadas na comunicação entre aplicações. Possibilita a aplicação do conceito de "troca de mensagens" explicado na Seção 2.8;
- **Flask - SQLAlchemy [45]:** extensão da plataforma *Flask* (utilizada no desenvolvimento de API's em Python) que possibilita a utilização do *SQLAlchemy*. O SQLAlchemy, por sua vez, é uma plataforma de mapeamento objeto-relacional SQL (ORM — *Object Relational Mapper*) capaz de facilitar a "tradução" de dados de uma tabela de um banco de dados relacional para um objeto a ser utilizado na programação em Python;
- **Banco de dados MySQL [46]:** banco de dados relacional utilizado no armazenamento dos dados coletados pelos dispositivos de sensoriamento;
- **Grafana [47]:** *software* de código aberto que consulta os dados armazenados no banco de dados e os apresenta na forma de gráficos interativos;

- **Docker [48]:** ferramenta utilizada na aplicação do conceito de "conteinerização" apresentado na Seção 2.7. Possibilita a construção e execução de contêineres Docker.

Estas ferramentas terão seus funcionamentos explicados nesta seção.

3.2.1 RabbitMQ

RabbitMQ [22] é um dos Mediadores de Mensagens (do inglês *Message Brokers*) de código aberto mais populares do mercado [22]. Entre suas funcionalidades, a aplicação traz:

- **Troca assíncrona de mensagens:** com suporte a múltiplos protocolos de trocas de mensagem, enfileiramento de mensagens, notificações de entrega, roteamento flexível para filas e múltiplos tipos de Permutador;
- **Troca de Mensagens entre linguagens:** suporte a troca de mensagens entre aplicações escritas em Python, .NET, Java, Go, Ruby, PHP, Javascript, entre outras;
- **Implantação em agrupamentos de servidores (*clusters*):** o RabbitMQ pode ser implementado sob arquitetura de forma distribuída para prover alta disponibilidade e taxa de transferência;
- **Autenticação via LDAP e TLS:** autenticação e autorização via protocolos LDAP e TLS;
- **Leve e de fácil implantação:** imagem disponível no *Docker Hub*, tornando a implantação de um contêiner RabbitMQ simples e fácil.

O RabbitMQ é um Mediador de Mensagens, ou seja, um gerenciador de mensagens entre aplicações. O funcionamento de um Mediador de Mensagens é melhor explicado na Seção 2.8. Basicamente, ele hospeda o Permutador e as filas. O Permutador é a entidade que faz o roteamento das mensagens geradas pelo Produtor para as devidas filas. Estas filas são então observadas pelo Consumidor até que tenham uma mensagem disponível para consumo. Uma vez disponível, a mensagem é então retirada da fila pelo Consumidor.

3.2.2 Flask - SQLAlchemy

Flask [24] é uma plataforma para a linguagem Python definida por Hunt-Walker [49] como "uma biblioteca de código que torna a vida do desenvolvedor mais fácil ao construir aplicações web confiáveis, escaláveis e sustentáveis". Isso é possível porque uma plataforma de desenvolvimento web (do inglês *framework*) lida com todas as questões de baixo nível como protocolos de comunicação e gerenciamento de *threads* [50]. Sendo assim, o uso do Flask possibilita a abstração de uma série de dificuldades que um desenvolvedor web enfrentaria. Neste projeto, todas

as aplicações desenvolvidas para recebimento de dados via HTTP foram desenvolvidas sobre esta plataforma.

Desenvolvido por Armin Ronacher, Flask é baseado no *toolkit Werkzeg WSGI* e no *Jinja2*, um mecanismo de *templates* [50]. Suas funções são:

- **WSGI:** o *Web Server Gateway Interface* é usado como padrão no desenvolvimento de aplicações web em Python. WSGI é uma especificação de uma interface comum entre web servers e aplicações web [50];
- **Werkzeug:** um kit de ferramentas WSGI que implementa requisições, objetos de resposta (às requisições) e funções utilitárias.
- **Jinja2:** um mecanismo de *templates* (do inglês *template engine*) em Python muito popular. Sua função é basicamente ligar um *template* web (uma espécie de "esqueleto" para uma página web) a uma fonte de dados que provê a informação a ser mostrada. Assim, a página web apresenta os dados da fonte dinamicamente.

O Flask é considerado um *"microframework"* porque ele mantém o núcleo da aplicação simples, mas escalável. Funcionalidades são adicionadas conforme extensões são introduzidas. Uma das extensões é o **Flask-SQLAlchemy**, que adiciona à aplicação suporte ao SQL Alchemy.

O SQL Alchemy é uma plataforma de mapeamento objeto-relacional SQL (ORM - *Object Relational Mapper*) de código aberto desenvolvida para a linguagem Python [51]. *ORM* é uma técnica que auxilia na utilização de dados de bancos relacionais em linguagens orientadas a objetos. Basicamente, o SQL Alchemy expande as funcionalidades do Flask para que a API consiga se comunicar com o banco de dados e fazer a conversão de dados de tabelas para objetos Python e vice-versa. A Figura 3.10 ilustra o funcionamento desta arquitetura. Navegadores, aplicativos móveis, e outras aplicações fazem requisições para a API escrita sobre o Flask, que por meio do SQLAlchemy, converte a linha da tabela no banco SQL em um objeto utitilizável em Python. Assim, a API consegue responder à requisição adequadamente. No caso de uma requisição POST ou UPDATE, o caminho contrário seria feito: a API recebe a requisição em um terminal de comunicação, então o SQLAlchemy faz as alterações necessárias para que ela possa ser gravada no banco de dados SQL.

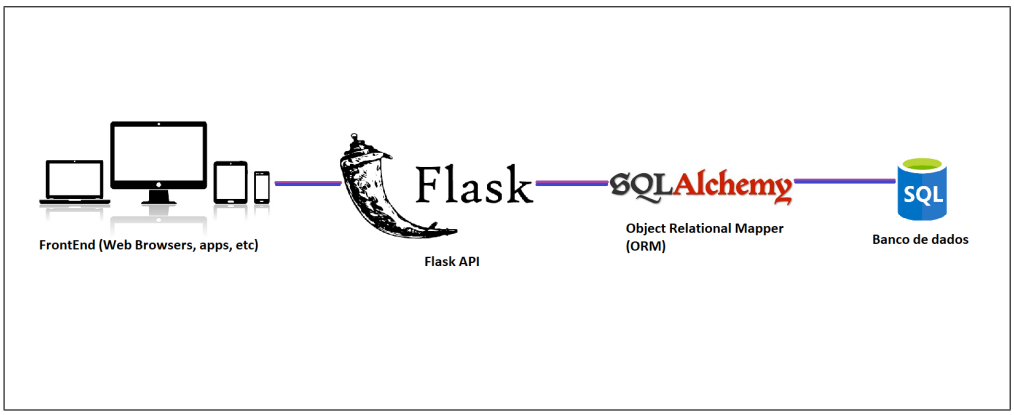

Figura 3.10: Caminho de comunicação com Flask-SQLAlchemy.

A Figura 3.10 ilustra o funcionamento desta arquitetura. Navegadores, aplicativos móveis, e outras aplicações fazem requisições para a API escrita no Flask, que por meio do SQL Alchemy, converte a linha da tabela no banco SQL em um objeto utitilizável em Python. Assim, a API consegue responder a requisição adequadamente. No caso de uma requisição POST ou UPDATE, o caminho contrário seria feito: a API recebe a requisição em um *endpoint*, então o SQL Alchemy faz as alterações necessárias para que ela possa ser gravada no banco SQL.

3.2.3 Banco de Dados - MySQL

Um banco de dados é uma coleção estruturada de dados. Os dados podem ser qualquer coisa, desde um simples número, uma galeria de fotos, ou as informações geradas por sensores em um sistema IoT. Para poder adicionar, acessar e processar dados armazenados em um banco de dados, faz-se necessário um sistema de gerenciamento de banco de dados como o *MySQL Server* [46], serviço que utiliza a linguagem SQL *(Structure Query Language)*. O MySQL é um dos sistemas de gerenciamento de banco de dados SQL de código aberto mais populares, desenvolvido e distribuído pela *Oracle Corporation*.[52] O sistema foi desenvolvido pela empresa *MySQL AB* e publicado, originalmente, em maio de 1995. Esta empresa foi adquirida pela *Sun Microsystems* e, em janeiro de 2010, a *Sun Microsystems* foi incorporada à *Oracle Corporation.*

O Banco de Dados MySQL é relacional, o que significa que ele armazena dados em tabelas separadas. Em um banco de dados relacional, cada linha na tabela é um registro com um ID exclusivo, atributo considerado *chave primária*. As colunas da tabela contêm atributos dos registros e cada registro tem um valor para cada atributo, facilitando as relações entre os dados.

Neste projeto, o banco de dados MySQL é utilizado no armazenamento e na organização dos dados gerados pelos dispositivos de sensoriamento. Graças ao formato relacional, o MySQL é capaz de organizar os dados referentes aos diferentes sensores em tabelas diferentes com colunas customizáveis em termos de nomes (das colunas) e tipos de dados.

3.2.4 Grafana

Grafana é uma solução de código aberto para análise de dados. Este *software* trabalha com métricas e gráficos que ajudam a interpretar os dados, além de atuar no monitoramento de aplicativos com a ajuda de telas de apresentação personalizáveis. Ele permite consultar, visualizar e criar alertas de acordo com as métricas estabelecidas pelo usuário, independentemente de onde os dados estejam armazenados. A ferramenta nos ajuda a estudar, analisar e monitorar dados ao longo de um período de tempo e tem compatibilidade com diversos bancos de dados, inclusive o *MySql* que foi utilizado neste projeto.

Apesar de não participar diretamente do funcionamento do sistema, o Grafana foi de extrema importância durante o desenvolvimento do projeto, pois possibilitou o acompanhamento em tempo real dos dados recebidos dos dispositivos de sensoriamento. Graças aos dados apresentados pelo Grafana, foi possível identificar padrões e tomar decisões quanto aos algoritmos de detecção.

3.2.5 Docker

Docker é uma ferramenta de código aberto projetada para tornar mais fácil criar, implantar e executar aplicativos usando o conceito de contêineres, explicado na Seção 2. Um contêiner pode ser comparado a uma máquina virtual. Ambos são utilizados com o objetivo de isolar as aplicações do sistema da máquina física que os hospeda. Máquinas virtuais, ou *Virtual Machines* (VMs), virtualizam o sistema operacional inteiro e alocam recursos de hardware da máquina hospedeira. Contêineres, por outro lado, fazem uso do núcleo do sistema operacional e dos recursos de *hardware* do sistema em que estão hospedados, sem fazer "reservas", o que torna o uso de recursos mais eficiente. Além de compartilharem os recursos de *hardware*, os recursos de sistema operacional (como bibliotecas) também são compartilhados, de forma que não há necessidade de subir um sistema operacional inteiro para cada contêiner (como acontece com VMs). A diferença entre máquinas virtuais e contêineres é ilustrada na Figura 3.11.

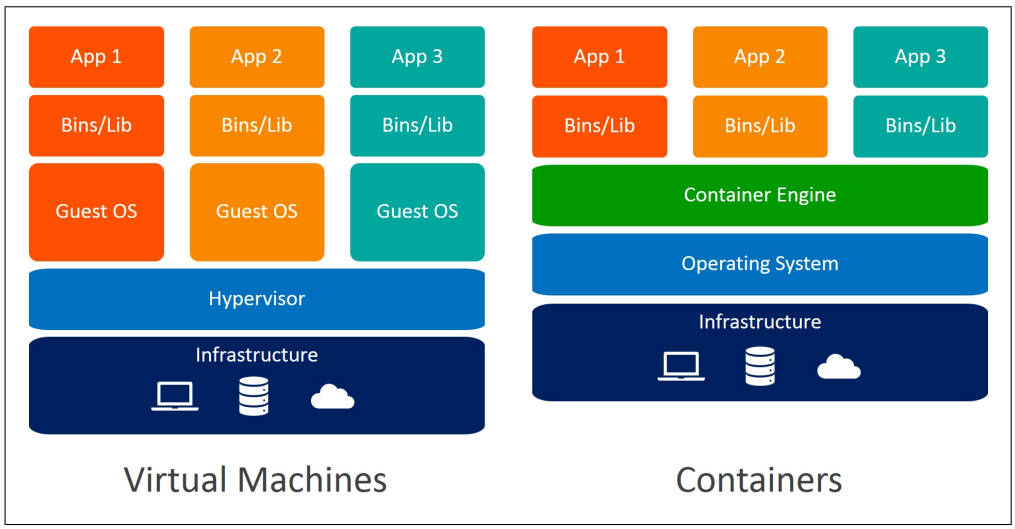

Figura 3.11: Comparativo: VMs vs Contêineres.[7]

Em resumo, contêineres são muito mais compactos em termos de espaço e mais eficientes em termos de recursos, mas ligeiramente "menos isolados", já que os processos executados nos contêineres são visíveis dentro da máquina hospedeira. O ganho de eficiência fica mais evidente quanto mais contêineres estiverem sendo executados, se comparados com VMs. Outra vantagem significativa do uso de contêineres é o tempo de inicialização, que leva poucos segundos.

Para este projeto, utilizou-se aliado ao Docker, o Docker Compose, que é um orquestrador para os contêineres Docker que facilita a implantação de aplicações formadas por múltiplos serviços em múltiplos contêineres. Para que o Compose orquestre esse sistema, utiliza-se um arquivo chamado docker-compose.yaml que é formatado de uma forma fácil de ler e entender. No arquivo YAML descreve-se a infraestrutura do sistema e como ela vai se comportar ao ser iniciada. Neste momento também é possível definir comportamentos dos contêineres caso algum deles encontre alguma falha. Pode-se também isolar essas aplicações em uma rede separada e definir os volumes onde as aplicações vão persistir seus dados. Ao final basta um comando `docker-compose up` para que todo o sistema esteja ligado e funcionando.

Há ainda o *Docker Hub* [53], que é o maior repositório de imagens de contêineres da Internet. Imagens Docker são "receitas" para a construção de contêineres que indicam tudo que a máquina hospedeira precisa para executar a aplicação. Desta forma, quando o contêiner é iniciado, o Docker prepara todos os requisitos especificados na imagem para que a aplicação seja iniciada assim que o contêiner estiver em funcionamento. Por exemplo, uma imagem que foi bastante usada neste projeto foi a do Python, que entrega todas as ferramentas e configurações necessárias para que em um comando, o Docker esteja executando nossa aplicação dentro de um contêiner.

3.2.6 Telegram

O Telegram é um aplicativo de mensagens com foco em velocidade e segurança, simples e grátis. O Telegram é baseado em um protocolo de dados personalizado chamado MTProto, construído pelo matemático Nikolai Durov. Neste projeto foi utilizado um Bot do Telegram para fazer a notificação dos usuários de eventos detectados pelo sistema. Basicamente, os Bots do Telegram são contas especiais que não exigem um número de telefone para serem utilizadas. Os usuários podem interagir com os bots de duas maneiras:

- Enviando mensagens e comandos para os bots abrindo um bate-papo com eles ou adicionando-os aos grupos.
- Enviando solicitações diretamente ao Bot, digitando @username do bot e uma consulta. Isso permite o envio de conteúdo dos Bots diretamente para qualquer chat, grupo ou canal.

Mensagens, comandos e solicitações enviadas pelo usuário são passadas para o *software* em execução no servidor da API do Sistema. Sendo assim, o servidor intermediário lida com toda a criptografia e comunicação com a API do Telegram.

Capítulo 4

Infraestrutura

A arquitetura do projeto depende de 4 componentes: o sistema embarcado da Bosch XDK, o sistema embarcado controlado pelo microcontrolador ESP32, um computador de propósito geral utilizado como *Gateway* e uma máquina virtual provida pela *Google Cloud Platform*[32].

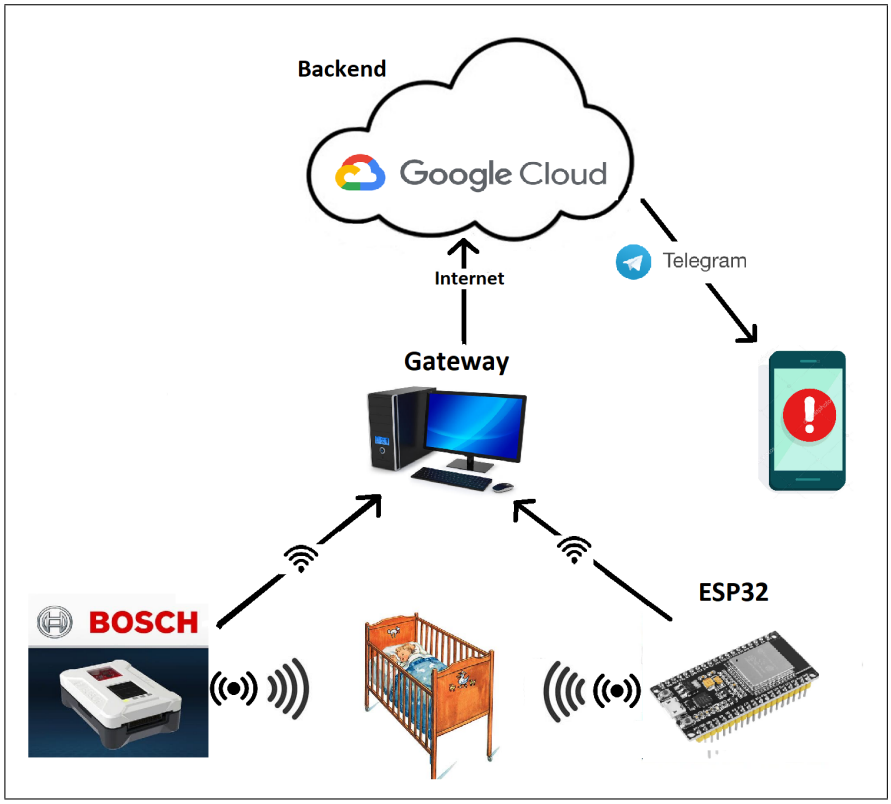

Figura 4.1: Arquitetura geral do projeto.

Na Figura 4.1 apresentada acima, estão ilustrados os 4 componentes-chave do sistema e o telefone celular de um responsável pelo bebê. As setas indicam o fluxo de dados, desde a coleta dos sensores até o processamento realizado na nuvem que culmina no envio de um alerta pelo

aplicativo Telegram.

De forma resumida, as funções de cada componente são:

- **Bosch XDK:** coleta dos dados referentes às condições do ambiente (temperatura, umidade do ar e iluminância) e envio dos mesmos ao servidor *Gateway*;
- **ESP32:** coleta dos dados referentes ao comportamento do bebê (áudio e movimento) e envio dos mesmos ao servidor *Gateway*;
- **Gateway:** enfileiramento de mensagens contendo dados provenientes de ambas as placas e envio ao *backend* na nuvem;
- **Backend:** servidor na nuvem responsável pelo armazenamento final dos dados e pela detecção de eventos e envio de alertas.

Neste capítulo, será detalhado o funcionamento de cada um dos dispositivos.

4.1 Bosch XDK

O kit XDK é encarregado da coleta dos dados referentes à temperatura do ambiente, à umidade do ar e à iluminância. O sensor aqui utilizado é o BME280, mencionado na Seção 3.1.1.2, apresentado como um "sensor ambiental" que capta informações de pressão atmosférica, temperatura do ambiente e umidade do ar. As informações de iluminância são captadas por um outro sensor, também embarcado no kit XDK, denominado de MAX44009, mencionado na Seção 3.1.1.3. Neste projeto, apenas as informações de temperatura e umidade do ar são coletadas do sensor BME280 e utilizadas, umas vez que a pressão atmosférica não deve alterar de forma significativa no ambiente onde o carrinho da criança se encontra.

Os valores dos sensores são lidos, gravados em variáveis globais e então introduzidos em um objeto JSON que é incorporado ao corpo da requisição HTTP POST enviada ao servidor *Gateway*. Este processo é repetido periodicamente (um envio por segundo), de forma a manter o banco de dados atualizado em relação às informações sobre o ambiente em que o bebê se encontra.

Nesta seção será explicado o desenvolvimento do código da aplicação executada pela Bosch XDK110, apresentando em detalhes as funções mais importantes. O código foi escrito em C utilizando o *XDK Workbench*, provido como um ambiente de desenvolvimento integrado (IDE, do inglês *Integrated Development Environment*) pela Bosch para a programação do kit XDK. A Figura 4.2 apresenta a tela típica da interface de desenvolvimento do *XDK Workbench*.

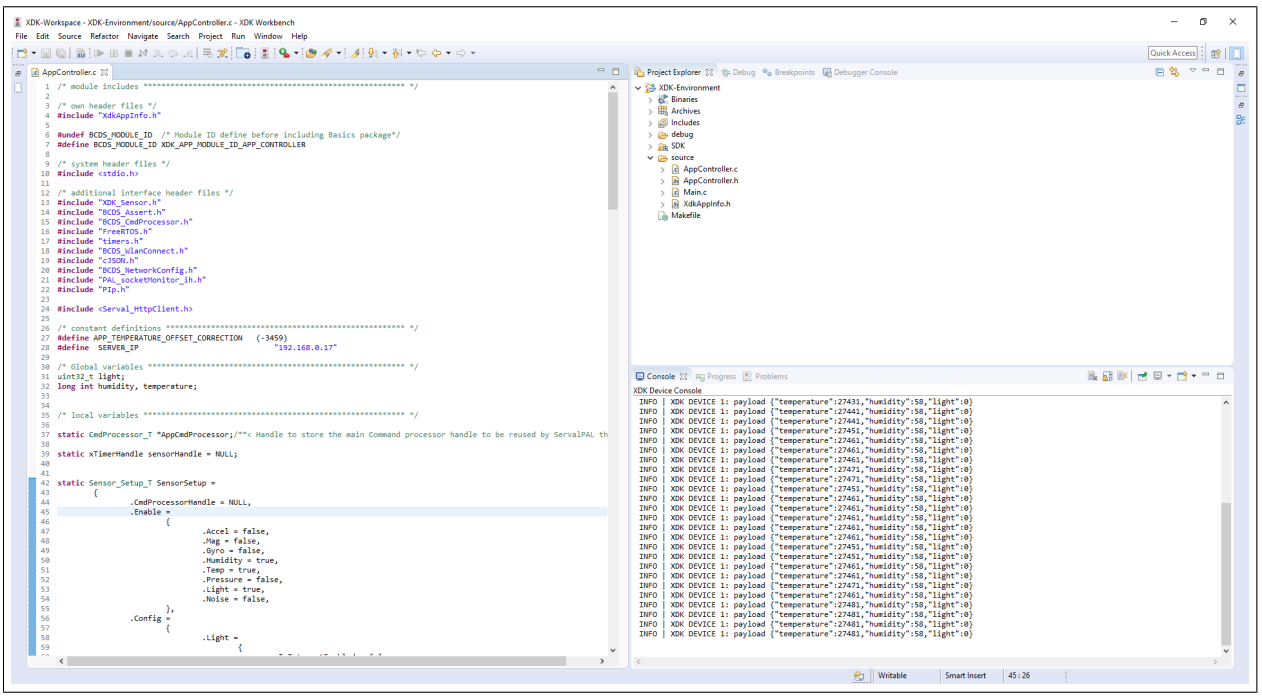

Figura 4.2: Interface XDK Workbench.

Com o *Workbench*, é possível fazer a compilação e a preparação do arquivo executável, bem como o armazenamento da aplicação na placa (salvando-a na memória *flash* do dispositivo) com apenas alguns cliques. Feito isso, é possível acompanhar a execução do programa pelo *"Device Console"* para realizar a depuração de erros.

A aplicação executada pelo kit deve fazer a coleta dos dados de temperatura, umidade do ar e iluminância (consultando os valores dos sensores BME280 e MAX44009), construir um documento JSON a partir de uma compilação destas informações, conectar-se à rede utilizando seu módulo Wi-Fi integrado e então fazer o envio dos dados ao *Gateway* por meio de uma requisição HTTP POST. Para cumprir estes objetivos, é utilizada uma série de bibliotecas inclusas na instalação do *XDK Workbench* — algumas de autoria da Bosch, outras desenvolvidas por terceiros e consolidadas na linguagem C ou em implementações de sistemas microprocessados (como as bibliotecas comumente utilizadas na programação de placas Arduino).

A coleta de todos os sensores foi projetada para ser realizada a cada segundo fazendo uso de um temporizador de *software* disponibilizado pelo próprio FreeRTOS, citado na Seção 2.3. O Sistema Operacional se responsabiliza pela contagem, e ao atingir 1 segundo, uma função *callback* é chamada para realizar a leitura dos sensores, guardar as informações em um documento JSON e enviá-lo ao servidor *Gateway* via HTTP POST. A aplicação será descrita em detalhes a seguir.

O código da função chamada pela interrupção é apresentado na Listagem 4.1 abaixo. A função `Sensor_GetData` que aparece na linha 10 é responsável pela coleta dos valores do sensor e

faz parte da biblioteca XDK_Sensor.h, uma das principais bibliotecas da placa XDK110. Baseado nos sensores habilitados previamente, ela realiza a coleta dos valores e os guarda dentro de um objeto da classe *Sensor_Value_T*, que possui os atributos *RH, Temp e Light*. Estes atributos são então evocados e guardados dentro de variáveis globais, conforme as linhas 13, 15 e 17.

```
1   static void AppControllerFire(void* pvParameters)
2   {
3       BCDS_UNUSED(pvParameters);
4
5       Retcode_T retcode = RETCODE_OK;
6       Sensor_Value_T sensorValue;
7
8       memset(&sensorValue, 0x00, sizeof(sensorValue));
9
10          retcode = Sensor_GetData(&sensorValue);
11          if (RETCODE_OK == retcode)
12          {
13              humidity = (long int) sensorValue.RH;
14
15              temperature = (long int) sensorValue.Temp;
16
17              light = sensorValue.Light;
18              }
19
20          if (RETCODE_OK != retcode)
21          {
22              Retcode_RaiseError(retcode);
23          }
24
25          Ip_Address_T destAddr;
26          PAL_getIpaddress((uint8_t*) SERVER_IP, &destAddr);
27          Ip_Port_T port = Ip_convertIntToPort(54321);
28
29          Msg_T* msg_prt;
30          HttpClient_initRequest(&destAddr, port, &msg_prt);
31          HttpMsg_setReqMethod(msg_prt, Http_Method_Post);
32          HttpMsg_setReqUrl(msg_prt,"/xdk");
33          HttpMsg_setHost(msg_prt, SERVER_IP);
34
35          HttpMsg_setContentType(msg_prt, Http_ContentType_App_Json);
36
37          Msg_prependPartFactory(msg_prt, &writeNextPartToBuffer);
38
39
40          static Callable_T sentCallable;
41          Callable_assign(&sentCallable, &onHTTPRequestSent);
42
```

```
43            HttpClient_pushRequest(msg_prt, &sentCallable, &
    onHTTPResponseReceived);
44     }
45
```

Listagem 4.1: Coleta e envio de dados na Bosch XDK.

A biblioteca HttpClient.h provê as funções necessárias para a configuração e envio de requisições HTTP. Após a definição de dados importantes para os cabeçalhos do *frame* HTTP, como IP de destino, porta de destino, tipo de requisição, URL e Content_Type (linhas 26, 27, 31, 32 e 35 da Listagem 4.1, respectivamente), a função `writeNextPartToBuffer` fica então encarregada de montar o objeto JSON que será enviado e então adicioná-lo ao *buffer* de envio.

```
1    retcode_t writeNextPartToBuffer(OutMsgSerializationHandover_T* handover)
2    {
3      static char  *payload;
4      static cJSON *JSON;
5
6      JSON = cJSON_CreateObject();
7
8      cJSON_AddItemToObject(JSON,"temperature",cJSON_CreateNumber(temperature
    ));
9      cJSON_AddItemToObject(JSON,"humidity",cJSON_CreateNumber(humidity));
10     cJSON_AddItemToObject(JSON,"light",cJSON_CreateNumber(light));
11     payload = cJSON_Print(JSON);
12     cJSON_Minify(payload);
13     printf("payload %s \r\n",payload);
14
15     cJSON_Delete(JSON);
16
17
18     uint16_t payloadLength = (uint16_t) strlen(payload);
19     uint16_t alreadySerialized = handover->offset;
20     uint16_t remainingLength = payloadLength - alreadySerialized;
21     uint16_t bytesToCopy;
22     retcode_t rc;
23     if (remainingLength <= handover->bufLen)
24     {
25       bytesToCopy = remainingLength;
26       rc = RC_OK;
27     }
28     else
29     {
30       bytesToCopy = handover->bufLen;
31       rc = RC_MSG_FACTORY_INCOMPLETE;
32     }
33
```

```
34      memcpy(handover->buf_ptr, payload + alreadySerialized, bytesToCopy);
35      handover->offset = alreadySerialized + bytesToCopy;
36      handover->len = bytesToCopy;
37      free(payload);
38      return rc;
39    }
40
```

Listagem 4.2: Montagem do Objeto JSON a ser enviado.

A função `writeNextPartToBuffer` é apresentada na Listagem 4.2. Utilizando a biblioteca da linguagem C conhecida como *cJson.h*, declara-se um objeto cJSON na linha 4. Como as três variáveis globais *temperature, humidity e light* são inteiros, utiliza-se a função *cJSON_CreateNumber* para associar os valores de temperatura, umidade e luminosidade às chaves *"temperature", "humidity"e "light"*. Nas mesmas linhas, estes dados são incorporados ao objeto JSON por meio da função *cJSON_AddItemToObject*. A função *cJSON_Print* é chamada na linha 11 para fazer a renderização do objeto em texto e guardá-lo na variável *payload*. Esta variável é então escrita no *buffer* de envio na linha 34. Alguns exemplos de *payloads* são apresentados na Figura 4.3.

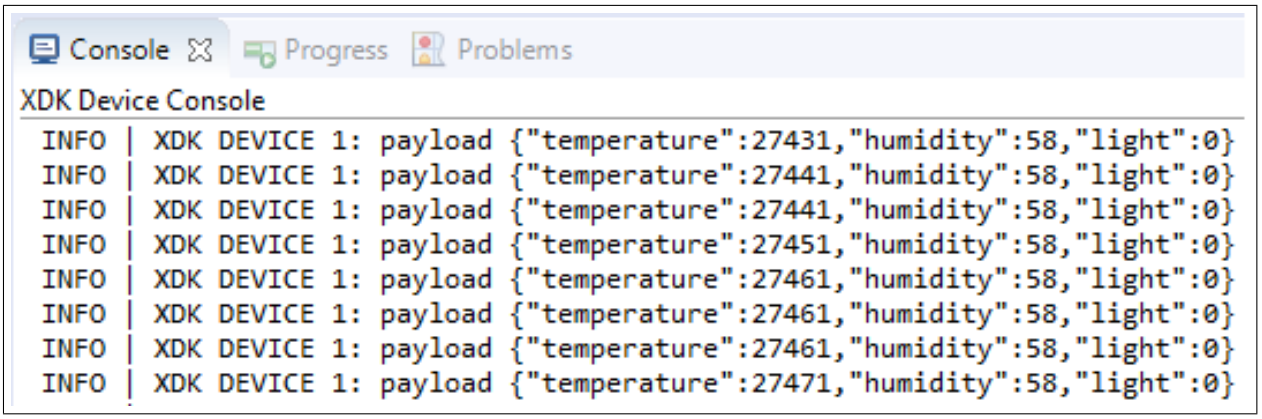

Figura 4.3: Exemplos de payloads enviados pela Bosch XDK.

A comunicação entre a placa e o *Gateway* é feita pela rede local configurada na residência do usuário. A placa conta com um módulo Wi-Fi com suporte aos padrões 802.11b/g/n do IEEE, funcionando na faixa de 2.4 GHz. Fazendo uso deste módulo, a placa se conecta à rede definida no código da aplicação (SSID e senha), define um endereço IP via DHCP e então passa a transmitir os dados. O endereçamento de destino (do *Gateway*) é configurado na aplicação. Na Figura 4.4, pode-se observar um pacote enviado pelo kit XDK ao *Gateway*, capturado pelo *software* farejador de pacotes *Wireshark* [54]. Nos cabeçalhos do pacote, pode-se notar os protocolos envolvidos, em especial o HTTP como protocolo da camada de aplicação e TCP como protocolo da camada de transporte, além dos endereços e portas utilizadas pelo kit e pelo *Gateway* na comunicação.

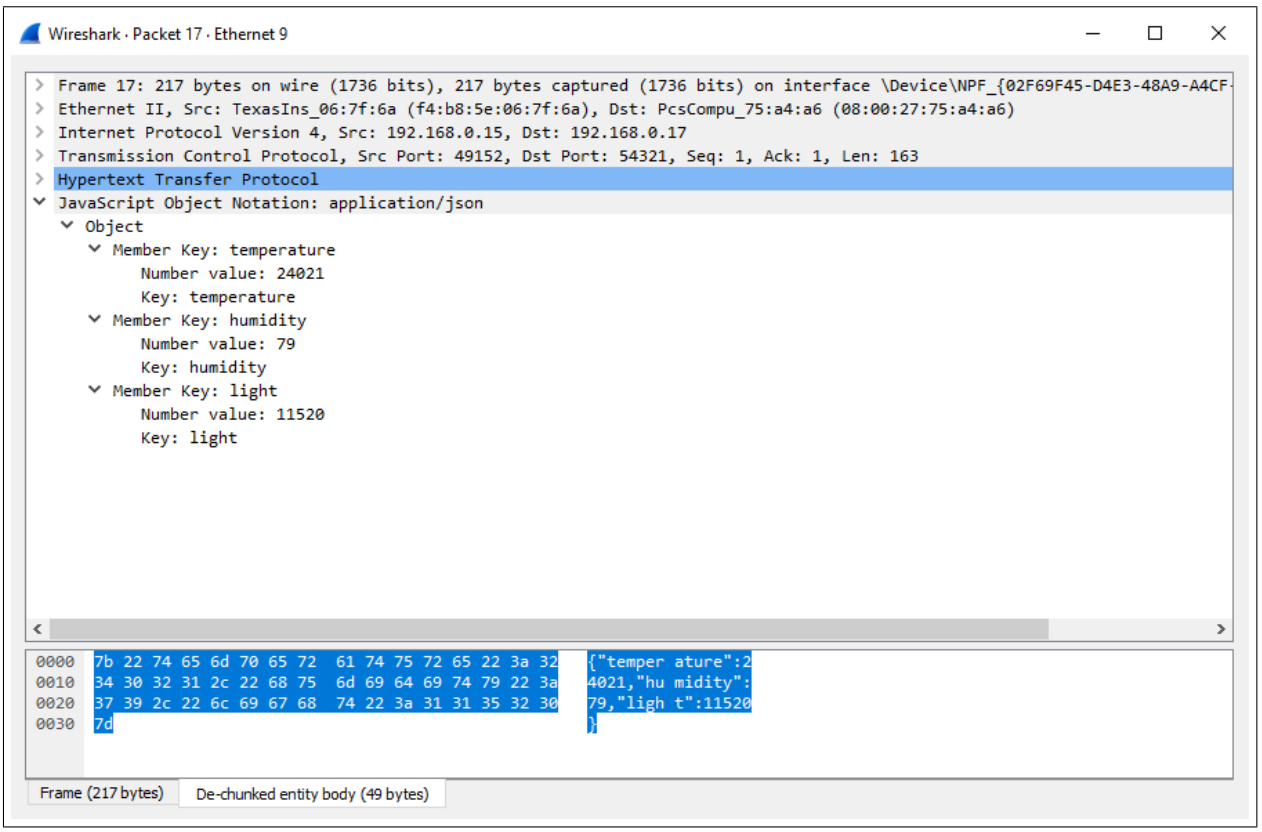

Figura 4.4: Pacote enviado do kit Bosch XDK ao Gateway.

4.2 Microcontrolador ESP32

O microcontrolador ESP32 é encarregado pela coleta dos dados referentes a áudio e a movimento. Os sensores aqui utilizados são o Sensor Acústico — Microfone MAX9814, mencionado na Seção 3.1.2.2 e o Sensor de Movimento Infravermelho Passivo — PIR, mencionado na Seção 3.1.2.3. Os valores dos sensores são lidos, e então introduzidos em um objeto JSON que é incorporado ao corpo da requisição HTTP POST enviada ao servidor *Gateway*. Este processo é repetido periodicamente (um envio por segundo).

Para transferir o código para o microcontrolador utilizou-se o *Arduino Integrated Development Environment(IDE)* [55]. Essa IDE contém um editor de texto para escrever código, uma área de mensagem, um console de texto, uma barra de ferramentas e uma série de menus. Ele se conecta ao microcontrolador ESP32 para transferir programas e receber informações sobre a execução da aplicação em seu *Serial Monitor*, utilizado na depuração de erros. Na Figura 4.5 pode-se observar a interface da IDE.

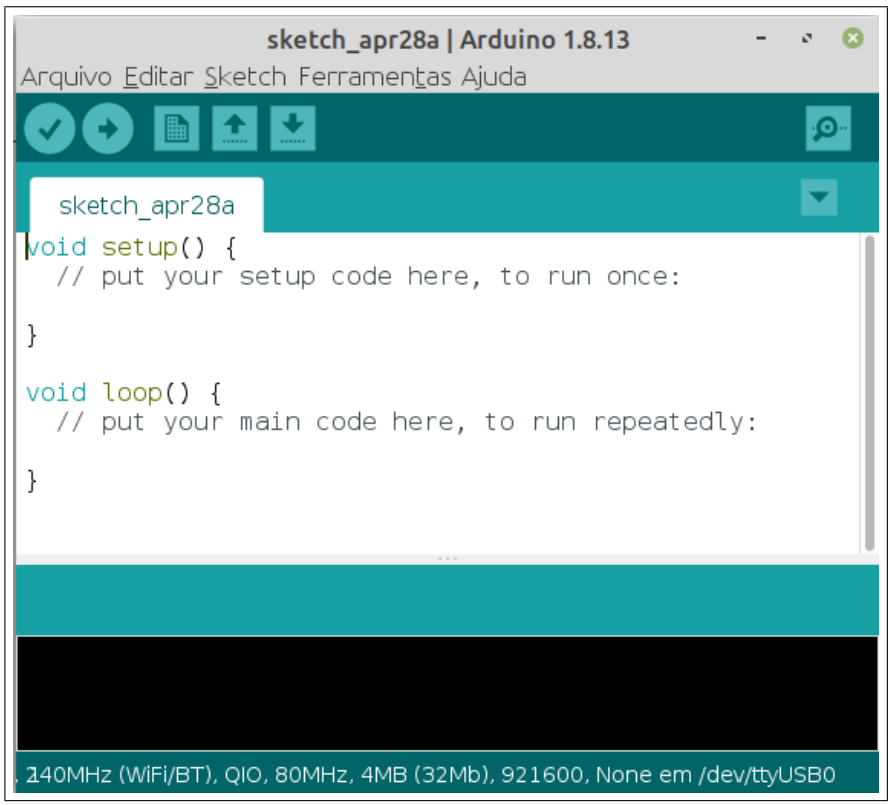

Figura 4.5: Novo projeto em branco criado pelo Arduino IDE com suas principais funções: `setup()` e `loop()`.

Tendo em vista que o microcontrolador utilizado neste projeto foi o ESP32, houve a necessidade de uma configuração específica na IDE. Há um recurso adicional para o Arduino IDE que permite programar o microcontrolador ESP32 usando a linguagem de programação C. Esse complemento pode ser obtido através do sítio da *Espressif Systems* [21], de modo que logo após a URL ser adicionada, pode-se encontrar no Gerenciador de Bibliotecas a biblioteca do microcontrolador ESP32. Feita a instalação da biblioteca, a IDE já pode comunicar-se com a placa utilizando a porta serial do computador.

Para a comunicação do microcontrolador ESP32 com a rede local, utilizou-se a biblioteca WiFi.h [56], que torna possível a utilização do Wi-Fi com suporte aos padrões IEEE 802.11 b/g/n, na faixa de 2.4 GHz. Essa biblioteca permite que a placa autentique-se e conecte-se à rede. Com isso, a placa pode servir como um servidor, que aceita conexões de entrada, ou como cliente que faz conexões de saída.

A coleta dos dados no microcontrolador ESP32 é feita 50 vezes por segundo, fazendo uso da função *delay()*. Ela aceita um único parâmetro do tipo **int** como argumento, que representa o tempo em milissegundos que o programa deve esperar até passar para a próxima linha de código. Portanto, o atraso entre cada iteração no projeto, para os dados de movimento e áudio foi de 20 milissegundos, e, após 50 iterações, o pacote de dados no formato JSON é enviado. O número de 50 amostras por segundo foi definido desta forma devido à necessidade da coleta de níveis de

ruído em uma frequência razoável para diminuir a probabilidade de perda de registro de alguma informação relevante, além do período de 20 milissegundos completar 1 segundo ao fim das 50 iterações sem deixar tempo ocioso.

O código carregado no microcontrolador ESP32 pode ser visto na Listagem 4.3. Na programação pela Arduino IDE, costuma-se definir, além de variáveis globais, funções e constantes, uma função `setup()` e a função `loop()`. Quando o código é compilado, a IDE utiliza uma função `main()` definida em uma de suas principais bibliotecas, que chama a função `setup()`, e depois a função `loop()` dentro de um *loop* infinito. A função `setup()`, linha 13, é encarregada pelas inicializações e configurações necessárias, aqui trata-se de configurar a conexão com a rede local e, então, repete-se a função `loop()`, linha 27, indefinidamente. Para a captura e envio dos dados trabalha-se com duas funções: uma delas é a função `loop()` e a outra é a função `postDataToServer()`, na linha 48, que recebe uma *String*. A função `loop()` é responsável pela construção da mensagem em formato JSON, fazendo uso da biblioteca ArduinoJson.h [57], dentro de um *loop* ***FOR***, linha 33. A função `postDataToServer()` consequentemente é acionada a cada 1 segundo, para fazer o envio das 50 amostras coletadas ao servidor *Gateway*. Todo esse processo é repetido indefinidamente, com o objetivo de estar sempre encaminhando os dados para o monitoramento do bebê.

```
1   #include <WiFi.h>
2   #include <ArduinoJson.h>
3   #include <HTTPClient.h>
4
5   const char *ssid = "Lucas_2G";
6   const char *password = "**********";
7   const char *serverName = "http://192.168.1.7:54321/esp32";
8
9   //  GPIOs
10  int movement_pin = 16;
11  int audio_pin = 35;
12
13  void setup() {
14  Serial.begin(115200);
15  WiFi.begin(ssid, password);
16
17  while (WiFi.status() != WL_CONNECTED)
18  {
19      delay(500);
20      Serial.println("Connecting to WiFi..");
21  }
22  Serial.println("Connected to the WiFi network");
23
24  pinMode(movement_pin, INPUT);
25  }
26
```

```
27    void loop() {
28
29      StaticJsonDocument<2048> jsonDoc;
30      JsonArray array_sound      = jsonDoc.createNestedArray("sound");
31      JsonArray array_movement   = jsonDoc.createNestedArray("movement");
32
33      for( int a = 0; a < 50; a++ ){
34        sensor_audio    = analogRead(audio_pin);
35        sensor_movement = digitalRead(movement_pin);
36        array_sound.add(sensor_audio);
37        array_movement.add(sensor_movement);
38        delay(20);
39      }
40
41      String json_serial;
42      serializeJson(jsonDoc, json_serial);
43
44      postDataToServer(json_serial);
45
46    }
47
48    void postDataToServer(String sensors) {
49
50      HTTPClient http;
51
52      http.begin(serverName);
53
54      http.addHeader("Content-Type", "application/json");
55
56      int httpResponseCode = http.POST(sensors);
57
58      if(httpResponseCode>0){
59        String response = http.getString();
60        Serial.println(httpResponseCode);
61        Serial.println(response);
62      }
63    }
64
```

Listagem 4.3: Montagem do Objeto JSON a ser enviado.

Na Figura 4.6, pode-se observar um pacote enviado pelo microcontrolador ESP32 ao *Gateway*, capturado pelo *software* farejador de pacotes *Wireshark* [54]. Nos cabeçalhos do pacote, pode-se notar os protocolos envolvidos, em especial o HTTP como protocolo da camada de aplicação e TCP como protocolo da camada de transporte, além dos endereços e portas utilizadas pelo microcontrolador e pelo *Gateway* na comunicação, pode-se notar especificamente o endereço IP

`(192.168.1.7)` utilizado pelo servidor *Gateway* e a porta `54321` utilizada pela aplicação no servidor *Gateway*, conforme a linha 7 da Listagem 4.3. Também nota-se a mensagem em formato JSON, que encaminha os vetores com dados de som e movimento, respectivamente.

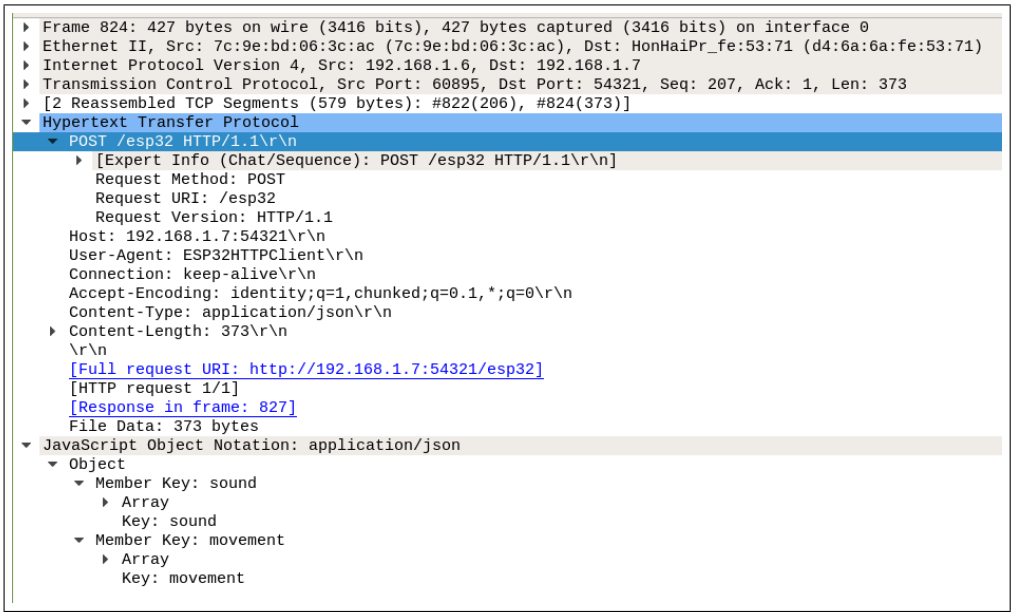

Figura 4.6: Pacote enviado do microcontrolador ESP32 ao gateway.

4.3 Gateway

O *Gateway* é a entidade responsável pelo interfaceamento entre o domínio sensorial (composto pelos sistemas microcontrolados e seus sensores) e o domínio de rede (onde se situa a Internet). Para facilitar a implementação desta entidade, foi aplicado o conceito de Conteinerização (consultar a Seção 2.7) por meio do uso do pacote de programas Docker[48], mencionado na Seção 3.2.5. O Docker possibilita a separação das funcionalidades do *Gateway* entre os contêineres cons, prod e rabbitmq. A máquina utilizada como *Gateway* neste projeto conta com a seguinte especificação:

- **Processador Intel Core i5-7300HQ CPU @ 2.50GHz**
- **8 GB de memória RAM**
- **Sistema Operacional Linux Mint 19.3 - Cinnamon**

Esta entidade foi projetada de acordo com a Figura 4.7 apresentada abaixo, onde estão representados os contêineres Docker. O Docker Compose[58], também citado na Seção 3.2.5, possibilita a descrição de vários **serviços** — basicamente aplicações, ou partes de aplicações, a serem executadas em contêineres — em um único arquivo, de forma que quando são inicializados, o

Docker oferece a opção de executá-los em contêineres separados ou não. Por padrão, os serviços são executados em contêineres separados. Os três serviços estão definidos em um arquivo docker-compose.yml. São eles: *prod, cons e rabbitmq*, os quais consistem no Produtor, no Consumidor e no Mediador de Mensagens, respectivamente (consultar a Seção 2.8).

O Produtor produz as mensagens a partir dos dados que recebe das placas em sua API e as publica para o Permutador que reside no serviço RabbitMQ. O Mediador de Mensagens hospeda o Permutador — que tem como função rotear as mensagens de acordo com suas chaves de roteamento para as filas correspondentes — e as filas propriamente ditas. Por fim, o Consumidor é responsável por acompanhar atualizações das filas e consumir as mensagens que se tornam disponíveis, o que significa enviá-las para processamento na nuvem.

O conteúdo do arquivo de configuração dos serviços é apresentado no Apêndice A.1. A versão do *Docker Engine* utilizada foi a 20.10.5, com o *Docker Compose* na versão 1.28.6.

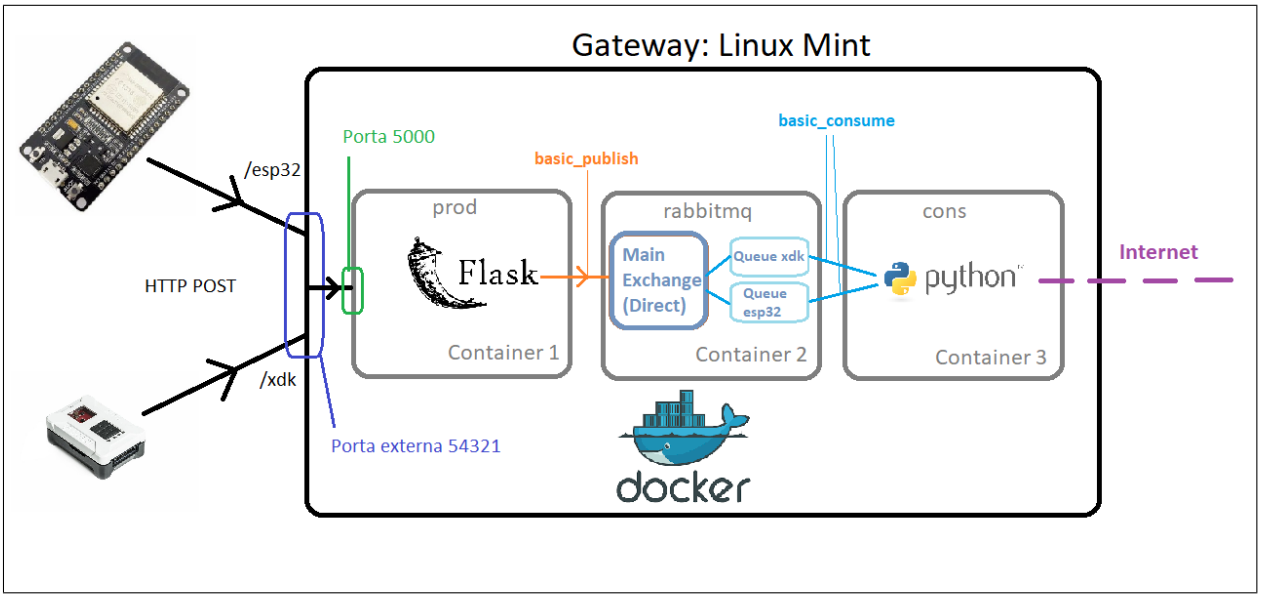

Figura 4.7: Arquitetura de contêineres Docker utilizados no Gateway.

O serviço "prod" representa o *Produtor* da Figura 2.1 e tem a porta 54321 da máquina hospedeira, isto é, a máquina Linux que executa o Docker, mapeada à porta 5000 do contêiner. A porta 54321 foi escolhida arbitrariamente, enquanto a porta 5000 é a escutada pela Flask API por padrão. Este é o único contêiner que precisa expor uma porta, uma vez que recebe as requisições do ambiente externo ao Docker.

O serviço "rabbitmq" representa o Mediador de Mensagens, ou seja, o gerenciador de mensagens. Nele são hospedadas as filas de mensagens recebidas das placas para que sejam posteriormente consumidas pelo serviço "cons" e enviadas para processamento na nuvem por meio da Internet.

O serviço "cons" representa o *Consumidor*, ou seja, a entidade que "escuta" as filas até que

haja uma mensagem disponível para consumo. Quando a mensagem está pronta, ela é retirada da fila e enviada para a nuvem pela aplicação *Consumidor*. A seguir, será descrito o funcionamento de cada um dos serviços em mais detalhes.

4.3.1 Produtor

O contêiner *prod* — ou Produtor — consiste em uma API construída em Python, sobre a plataforma Flask, que disponibiliza terminais de comunicação para o recebimento de dados de ambas as placas.

O Docker Hub é a maior biblioteca e comunidade de imagens de contêineres Docker do mundo [53], sendo imagens basicamente *templates* com as informações necessárias para a criação de um contêiner Docker. Utiliza-se no contêiner prod a imagem *python:3.8.6-buster* do *Docker Hub*, que contém a versão 3.8.6 do Python já instalada no *Debian Buster*.

Ao carregar o contêiner, o Docker faz a instalação dos pacotes necessários para o funcionamento da API Flask que foi desenvolvida para o projeto. São eles: *flask, flask-API e pika*. As bibliotecas *flask e flask-API* fornecem funções básicas para o funcionamento de uma API, como as funções *request* para fazer requisições, a classe status para tratamento dos códigos de *status* de requisições, entre outras. A biblioteca *pika* provê as funções e classes necessárias para a utilização do protocolo de troca de mensagens AMQP 0-9-1 na comunicação com o Mediador de Mensagens RabbitMQ.

A API desenvolvida fornece dois terminais de comunicação: /xdk e /esp32. Estes terminais estão programados para publicarem o conteúdo da requisição (o objeto JSON) ao Permutador (entidade responsável pelo roteamento de mensagens para as filas do RabbitMQ) chamado de *main_ex* assim que receberem uma requisição POST. Estas publicações são feitas por meio da função `basic_publish` da biblioteca *pika*, onde devem ser especificados o nome do Permutador, a chave de roteamento e o corpo da mensagem. A função `basic_publish` pode ser observada no apêndice A.2. O funcionamento do Permutador e das chaves de roteamento (*routing keys* em inglês) é detalhado nas Seções 2.8 e 3.2.1.

Dependendo do terminal de comunicação, o Produtor "produz" a mensagem a partir do conteúdo do objeto JSON e a publica com a devida chave de roteamento. Assim, quando o *Gateway* recebe uma requisição HTTP POST da XDK, o seu conteúdo é publicado ao Permutador com a chave "xdk", possibilitando que a mensagem seja posicionada na fila correta, destinada a mensagens do kit XDK. O mesmo acontece com os dados recebidos do microcontrolador ESP32, utilizando a chave de roteamento "esp32". Isso é possível porque o Permutador utilizado é do tipo Direto (consultar a Seção 2.8), portanto quando a ligação é feita entre uma fila e o Permutador, uma chave de roteamento é associada e então utilizada no roteamento. As mensagens com chaves de roteamento que correspondem à fila são encaminhadas a ela e neste caso, tem-se uma

fila para a XDK e outra para o microcontrolador ESP32.

4.3.2 RabbitMQ

O contêiner *rabbitmq* executa a aplicação Mediador de Mensagens, isto é, a aplicação responsável pelo gerenciamento de mensagens por meio do Permutador e das duas filas associadas a ele, uma para o microcontrolador ESP32 e uma para o kit XDK. As mensagens destas filas são então processadas pelo Consumidor e seus conteúdos são transmitidos via Internet para processamento na nuvem.

Neste contêiner, graças ao *Docker* e ao *Docker Hub*, é possível que o serviço seja executado e utilizado na infraestrutura sem a necessidade de escrever uma linha de código adicional sequer. O *Docker Hub* disponibiliza uma imagem "rabbitmq", que fornece a aplicação *RabbitMQ* executada em um Ubuntu 18.04. Como a declaração dos Permutadores, das filas e das associações entre eles (chamadas de *binds* em inglês, como descrito na Seção 2.8) pode ser feita no Produtor ou no Consumidor, não há necessidade de desenvolvimento de código para utilizar este contêiner. Em outras palavras, a imagem provida pelo *Docker Hub* fornece a aplicação *RabbitMQ* já pronta para execução e sua configuração é feita no outros dois contêineres, portanto não há necessidade de desenvolvimento de código adicional. A utilização da imagem do *Docker Hub* pode ser observada no apêndice A.1.

4.3.3 Consumidor

O último contêiner, "cons", é responsável pela retirada das mensagens das filas e o envio delas via Internet para processamento na nuvem. A imagem do *Docker Hub* utilizada como base é a *python:3.8.6-buster*, assim como no serviço "prod". Os pacotes instalados quando o contêiner é iniciado são o *pika* e o *requests*. Como mencionado anteriormente, a biblioteca *pika* fornece funcionalidades que remetem ao uso do protocolo AMQP 0-9-1 para a comunicação com o serviço RabbitMQ executado no contêiner rabbitmq, e a biblioteca *requests* fornece funções para envio de requisições HTTP.

Sua função é executar uma aplicação escrita em Python que tem dois objetivos: processar as mensagens das filas associadas ao kit XDK e ao microcontrolador ESP32, respectivamente, e então enviar os dados para processamento na nuvem. O código desta aplicação consiste na declaração das filas e das associações entre elas e o Permutador (denominadas *binds* em inglês), na chamada de funções que processam o conteúdo destas filas, e na chamada de funções que enviam as requisições HTTP POST ao servidor na nuvem. Algumas declarações são feitas mais de uma vez em arquivos diferentes (isto é, no Produtor e no Consumidor), como por exemplo a declaração do objeto Permutador. A função faz a tentativa da criação da entidade, mas se ela já

existe, nada é executado. Algumas instâncias são declaradas mais de uma vez por precaução no caso de um contêiner ser iniciado antes de outro.

O código da aplicação executada no contêiner cons é apresentado no Apêndice A.3. Para fins didáticos, ele será apresentado de forma simplificada no pseudocódigo listado no Algoritmo 1 abaixo.

Algoritmo 1: Aplicação do Consumidor descrito como pseudocódigo.

início

importação de bibliotecas;
estabelecimento da conexão com o contêiner rabbitmq via AMQP 0-9-1;
declaração do Permutador;
declaração das filas: esp32 e xdk;
declaração das associações entre filas e Permutador (binds), associando uma chave de roteamento a cada uma;
chamada das funções que consomem as duas filas, passando o nome da fila e uma função de envio ao terminal de comunicação correto do servidor na nuvem baseado no dispositivo que originou os dados;
chamada da função que inicia o loop de consumo;

fim

Como pode-se observar no Algoritmo 1, o Permutador e as filas são declaradas para que então sejam associadas por meio de chaves de roteamento. O Permutador se atrela à fila de mensagens provenientes do kit XDK por meio de uma chave de roteamento "xdk" e também se atrela à fila de mensagens provenientes do microcontrolador ESP32 por meio de uma outra chave de roteamento "esp32". A função de consumo, característica do serviço Consumidor, é chamada indicando a fila que deve ser observada e qual função deve ser chamada quando uma mensagem estiver disponível. A função chamada utiliza o conteúdo da própria mensagem para incluí-la no corpo da requisição HTTP POST e enviá-la ao servidor na nuvem por meio do terminal de comunicação correspondente à fila: /xdk ou /esp32.

4.4 Processamento na nuvem

Os dados recebidos pelo *Gateway* são enviados para processamento na nuvem, via Internet. Para armazenamento e processamento dos dados coletados, utilizamos uma máquina virtual disponibilizada pela Google em sua plataforma conhecida como *Google Cloud Platform* (GCP). O serviço *Compute Engine* da GCP disponibiliza máquinas virtuais que podem ser executadas na infraestrutura distribuída da Google. O acesso é feito via interface Web como apresenta a Figura 4.8.

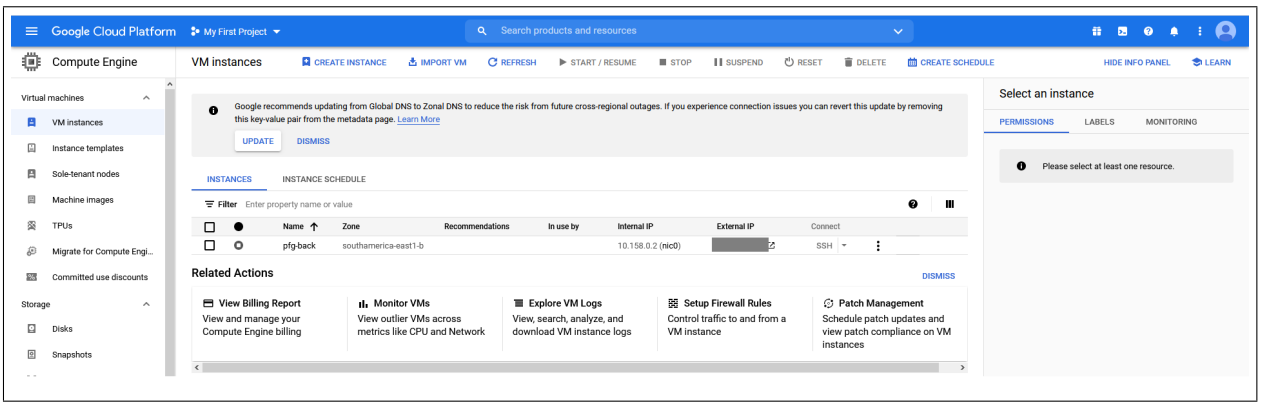

Figura 4.8: Interface Web da *Google Cloud Platform* (GCP).

Nesta simples interface, é possível especificar e iniciar uma máquina virtual com várias opções de configuração diferentes (sob custos diferentes). A máquina utilizada é uma das opções de máquinas de propósito geral da série E2, denominada de *e2-medium*. As máquinas da série E2 têm o custo otimizado, mas têm opções de uso de processadores com até 32 núcleos virtuais e 128 GB de memória RAM. A *e2-medium* utilizada neste projeto conta com 2 vCPUs e 4 GB de memória RAM. O sistema operacional instalado na máquina virtual é o Debian 10 (Buster) e a latência observada por meio de comando *ping* executado no computador *Gateway* foi de aproximadamente 35 ms (o servidor é hospedado em São Paulo).

Os custos dos serviços da GCP foram arcados com o programa para novos clientes que a Google proporciona, oferecendo US$ 300,00 para serem gastos em um período de 90 dias. Além disso, há opções de máquinas gratuitas com recursos mais limitados.

As configurações de *firewall* foram ajustadas para permitir o tráfego TCP proveniente do *Gateway*, que utilizava a porta 54322, além dos acessos remotos realizados via SSH pela porta padrão 22. Os acessos fazem uso de autenticação por chaves SSH para que se tenha mais segurança. Um IP externo fixo também foi configurado para que não fosse necessária a alteração do código da aplicação do *Gateway* que faz os envios para a *Google Cloud Platform* para processamento.

O subsistema de processamento de dados na nuvem foi implementado com uso do conceito de contêineres. Nele, são executados os contêineres denominados *app, mysql, grafana, cry, still, bad_env e bot*. Assim como no computador *Gateway*, o *Docker* e o *Docker Compose* são instalados para fazer a orquestração e a execução destes contêineres. As versões do Docker e do Docker Compose são as mesmas utilizadas no *Gateway*: 20.10.5 e 1.28.5, respectivamente. A Figura 4.9 ilustra a disposição dos contêineres e a comunicação entre eles.

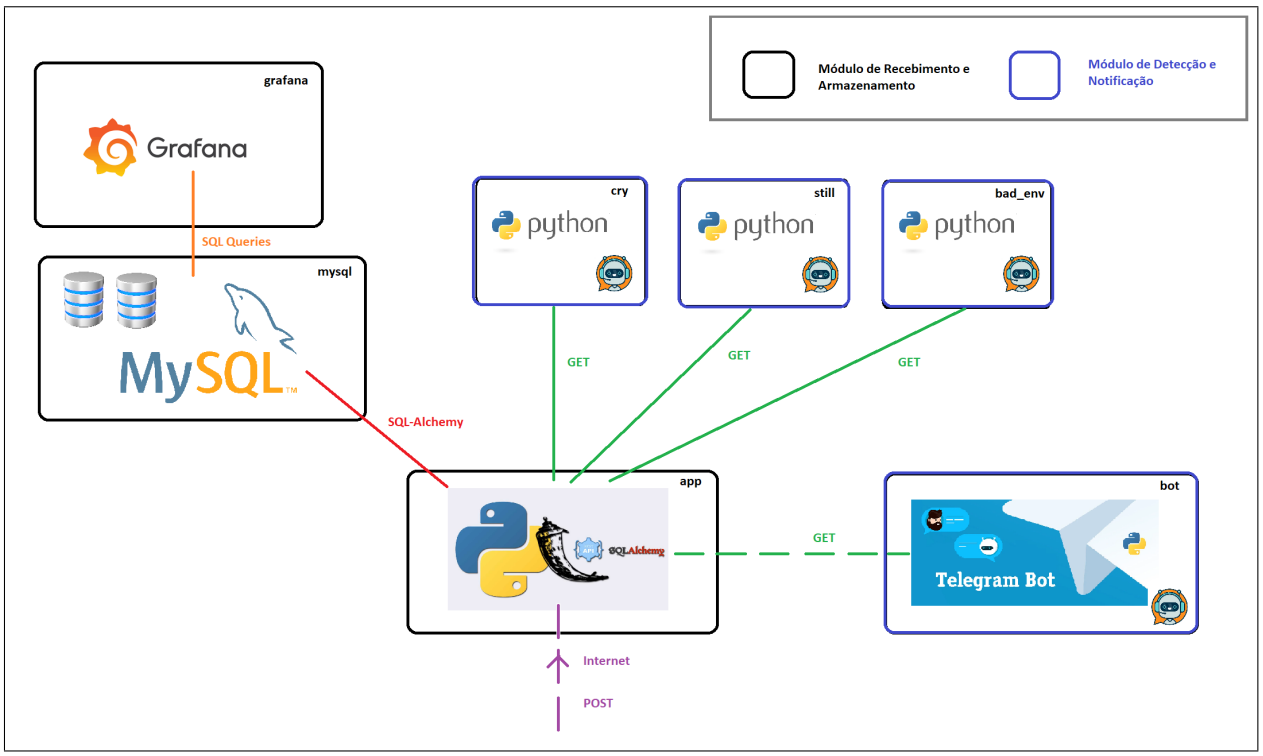

Figura 4.9: Arquitetura de contêineres Docker executados na máquina virtual da *Google Cloud Platform*.

A Figura 4.9 apresenta os contêineres executados na máquina virtual hospedada pelo serviço de nuvem da Google. O processamento é divido em:

- **Módulo de Recebimento e Armazenamento**

  Flask API: API de nossa autoria responsável pelo recebimento dos dados provenientes do *Gateway*;

  Banco de dados MySQL: banco de dados relacional utilizado no armazenamento de todas as informações dos sensores de ambos os dispositivos de sensoriamento;

  Servidor Grafana: realiza consultas diretamente no banco de dados para apresentar as tendências dos sinais captados pelos sensores de forma gráfica.

- **Módulo de Detecção e Notificação**

  cry: aplicação responsável pela detecção de eventos sonoros como choros, gritos e gemidos;

  still: aplicação responsável pela detecção de eventos relacionados a agitação excessiva ou ausência de movimento;

  bad_env: aplicação responsável pela detecção de condições de ambiente prejudiciais;

  bot: aplicação responsável por responder os usuários quando desejam ativamente consultar as condições ambientais do quarto.

O funcionamento geral de ambos os módulos, bem como o funcionamento específico de cada aplicação executada em cada contêiner, serão apresentados nas seções a seguir.

4.4.1 Módulo de Recebimento e Armazenamento

Este módulo é responsável pelo recebimento de dados dos sensores e armazenamento no banco de dados, principalmente. Dele fazem parte os contêineres que devem ser iniciados antes dos demais, pois sem a API e o banco de dados, as outras aplicações não funcionam. Além dos contêineres *app* e *mysql*, responsáveis pela API e pelo banco de dados, respectivamente, também faz parte deste módulo o contêiner *grafana*, que provê uma interface gráfica que apresenta as informações armazenadas no banco na forma de gráficos interativos. Isso permite que os sinais capturados pelos sensores sejam acompanhados pelo time de desenvolvimento em tempo real, possibilitando que ajustes e correções no algoritmo de detecção sejam feitos.

O apêndice B.1 apresenta o arquivo *docker-compose.yml* do Módulo de Recebimento e Armazenamento, onde são definidos os 3 serviços executados nos 3 contêineres, além da rede virtual que os conecta (criada e gerenciada pelo Docker), volumes utilizados para a persistência dos dados (significa que os dados são mantidos após a finalização do contêiner) e alguns outros parâmetros. Graças a este arquivo, todo o módulo pode ser iniciado em um terminal com apenas o comando `docker-compose up`.

4.4.1.1 Contêiner app

O contêiner *app* executa uma API escrita em Python sobre a plataforma Flask, assim como o contêiner *prod* do *Gateway*. Sua função é receber o tráfego de dados do *Gateway*, armazená-los no banco de dados fazendo uso da ferramenta *SQL-Alchemy* e prover os dados necessários para as aplicações responsáveis pela detecção de eventos e envio de notificações.

Tecnicamente, a aplicação executada no contêiner *app* faz parte de ambos os módulos simultaneamente (Módulo de Recebimento e Armazenamento e Módulo de Detecção e Notificação) uma vez que além de atuar na funcionalidade de recebimento e armazenamento, também atua no fornecimento de dados para a detecção de eventos. Este contêiner só é descrito no Módulo de Recebimento e Armazenamento porque ele é essencial para o funcionamento de todo o sistema e deve ser iniciado junto com a aplicação do banco de dados.

O Apêndice B.1 apresenta o arquivo *docker-compose.yml* utilizado na descrição das configurações feitas no ambiente Docker ao iniciar os contêineres. Primeiramente, o contêiner *app* tem a sua porta 5000 (padrão em aplicações Flask) mapeada para a porta 54322 da máquina hospedeira. Esta porta (54322) é exposta à Internet para que o *Gateway* consiga fazer os seus envios. O serviço também é configurado para reiniciar o contêiner no caso de terminação por qualquer causa

que não seja a intervenção manual de um usuário. Também é explicitado ao orquestrador Docker Compose que o serviço *mysql* deve estar em execução antes que o serviço *app* seja iniciado.

Seu código é dividido em uma série de arquivos com funções diferentes:

- **__init__.py:** inicialização da API, estabelecimento da comunicação com o banco de dados via SQLAlchemy;
- **model.py:** definição dos modelos que definirão as tabelas do banco posteriormente. Aqui são definidos as tabelas e as suas colunas (nomes e tipos de dados);
- **serializer.py:** faz a serialização das classes de objetos recebidos em formato JSON, definindo seus campos para que o SQLAlchemy consiga fazer a "conversão" do objeto para uma linha em uma das tabelas do banco;
- **xdk.py:** provê terminais de comunicação para o recebimento dos dados da placa XDK (requisições HTTP método POST) e para o envio de dados de luz, umidade e temperatura para as aplicações de detecção (requisições HTTP método GET);
- **esp32.py:** provê terminais de comunicação para o recebimento dos dados do microcontrolador ESP32 (requisições HTTP método POST) e para o envio de dados de som e movimento para as aplicações de detecção (requisições HTTP método GET);

Os terminais disponíveis para consulta de dados são:

ESP32:

- **/sound - método GET:** retorna os últimos 100 pontos da tabela *sound*, que guarda os registros de som. Como são enviados 50 pontos por segundo, este terminal retorna os últimos 2 segundos;
- **/movement - método GET:** retorna os últimos 100 pontos da tabela *movement*, que guarda os registros de movimento. Assim como no caso acima, ele retorna informações dos últimos 2 segundos;
- **/movement/still - método GET:** retorna os últimos 500 pontos da tabela *movement*, cobrindo 10 segundos;
- **/esp32 - método POST:** recebe os dados do microcontrolador ESP32 (na verdade sob o endereço de origem do *Gateway*) e os armazena nas tabelas *sound* e *movement*.

XDK:

- **/light - método GET:** retorna os últimos 20 pontos da tabela *light* que guarda os dados referentes à luminosidade. Como a XDK envia 1 valor de cada sensor por segundo, este terminal retorna informações dos últimos 20 segundos;
- **/light/now - método GET:** retorna o último registro da tabela *light*;
- **/humidity- método GET:** retorna os últimos 20 pontos da tabela *humidity*, que guarda os dados referentes à umidade. Isso representa 20 segundos de informações;
- **/humidity/now - método GET:** retorna o último registro da tabela *humidity*;
- **/temperature- método GET:** retorna os últimos 20 pontos da tabela *temperature*, que guarda os dados referentes à temperatura. Assim como os demais, 20 pontos representam 20 segundos de informações;
- **/temperature/now - método GET:** retorna o último registro da tabela *temperature*;
- **/xdk - método POST:** recebe os dados da Bosch XDK (na verdade sob o endereço de origem do *Gateway* e os armazena nas tabelas *light*, *humidity* e *temperature*.

Um exemplo de requisição GET enviado ao servidor é apresentado na Figura 4.10. A resposta a esta requisição é apresentada na Figura 4.11. Ambas Figuras 4.10 e 4.11 apresentam pacotes HTTP capturados pelo *Software* farejador de pacotes *Wireshark* [54]. O conteúdo do objeto JSON enviado na resposta será explicado na próxima seção, quando tratarmos do banco de dados.

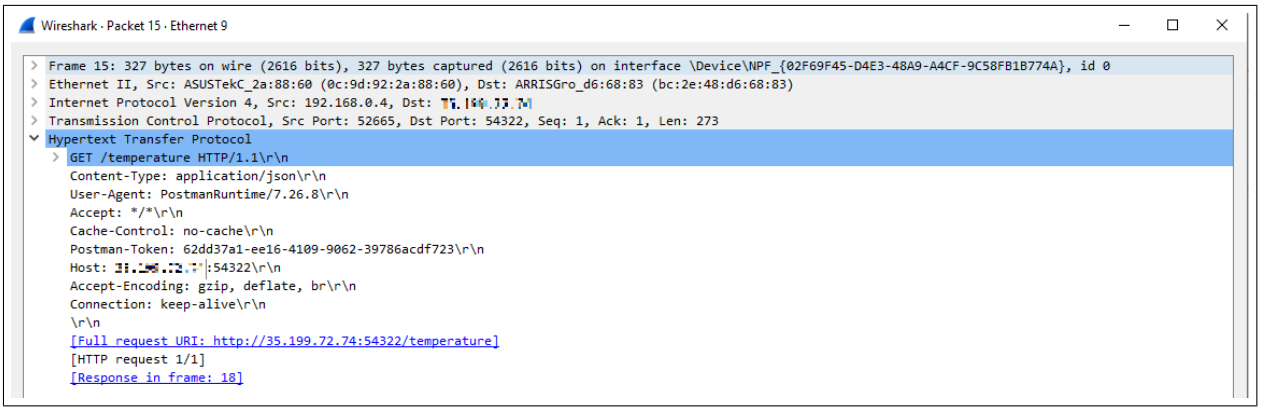

Figura 4.10: Pacote capturado contendo a requisição HTTP GET feita ao terminal /temperature.

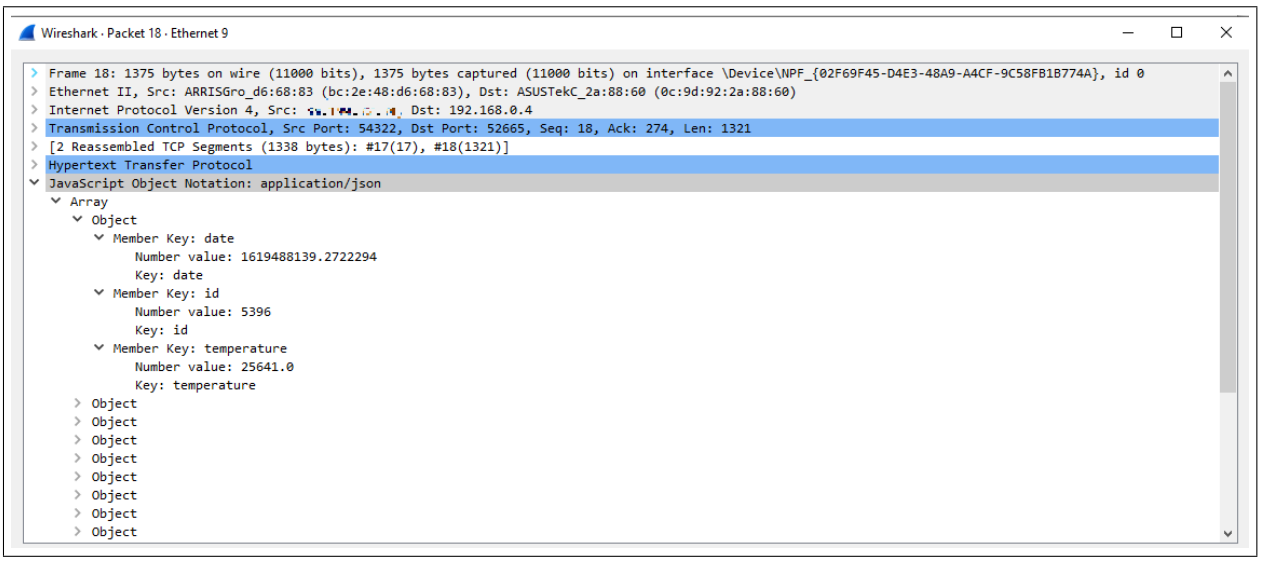

Figura 4.11: Captura do pacote de resposta à requisição da Figura 4.10.

Este contêiner precisa que o contêiner *mysql* esteja em execução para que a conexão com o banco seja estabelecida. O *app* e o *grafana* dependem do *mysql*, mas não são dependentes entre si.

4.4.1.2 Contêiner mysql

O contêiner *mysql* é responsável por hospedar o banco de dados MySQL utilizado para o armazenamento dos dados coletados pelo sistema. Todo o sistema depende do funcionamento do banco de dados para operar e por isso ele deve ser iniciado primeiro.

A imagem "mysql" disponível no *Docker Hub* entrega todas as ferramentas de sistema e configurações necessárias, além de já fazer a instalação da própria aplicação MySQL. O Apêndice B.1 mostra o arquivo *docker-compose.yml* utilizado na configuração do serviço mysql. Nele, um *contexto* (termo utilizado pela própria ferramenta *Docker Compose*) é provido para que o *Docker Compose* procure um arquivo *Dockerfile* no caminho especificado. Este arquivo, por sua vez, evoca a imagem *mysql* do *Docker Hub* utilizando a versão mais recente possível da aplicação, que no momento do desenvolvimento deste projeto é o MySQL 8.0.24. Além disso, o *Dockerfile* também configura uma senha customizada para o usuário *root* e executa, ao iniciar o contêiner, alguns comandos que fazem ajustes no fuso-horário do sistema operacional. Além da definição do *contexto*, o arquivo *docker-compose.yml* também declara o volume *dbdata*. Volumes no Docker representam armazenamento compartilhado entre máquina hospedeira e contêiner, o que possibilita que os dados gravados durante a execução do contêiner não sejam perdidos no momento de sua terminação.

A criação da base de dados e suas tabelas é feita na verdade pelo Flask-SQLAlchemy. A base utilizada se chama *crud*, que representa *Create, Read, Update and Delete*, as quatro operações

básicas utilizadas em bancos de dados relacionais. Dentro desta base estão nossas tabelas: *sound, movement, light, temperature* e *humidity*. Todas as tabelas têm apenas 3 colunas: uma que guarda o valor registrado, uma com o horário da captura e uma com o identificador do registro. A base *crud* tem suas tabelas ilustradas na Figura 4.12. Cada tabela tem uma coluna *id*, que é o identificador automaticamente incrementado em cada registro, uma coluna com o mesmo nome da tabela que guarda o valor registrado pelo sensor e uma coluna *date*, que guarda o registro de tempo do registro na forma de um decimal de 17 algarismos totais e 7 casas depois da vírgula de precisão.

Este registro de tempo é guardado no formato *Unix Epoch time*, que registra os segundos passados desde 1º de Janeiro de 1970 (UTC). A precisão decimal foi aumentada pois os registros são obtidos a cada 20 ms no caso do microcontrolador ESP32.

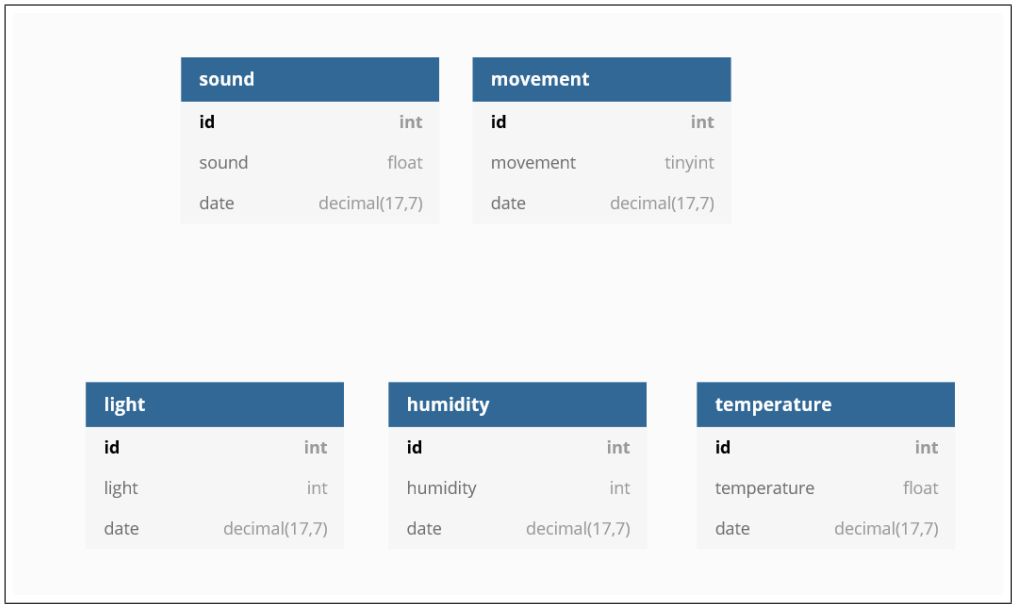

Figura 4.12: Diagrama da base *crud* e suas tabelas. Abaixo do nome de cada tabela, são apresentados os nomes das colunas e o tipo de dado que elas guardam.

4.4.1.3 Contêiner grafana

O serviço do Grafana é responsável pela geração de gráficos em tempo real a partir dos dados guardados no banco. Ele nos permite acompanhar as alterações nos valores dos sensores por meio de uma interface gráfica interativa. Com o Grafana, é possível criar e organizar telas de apresentação com gráficos interativos, tendo a opção de atualização automática ou não.

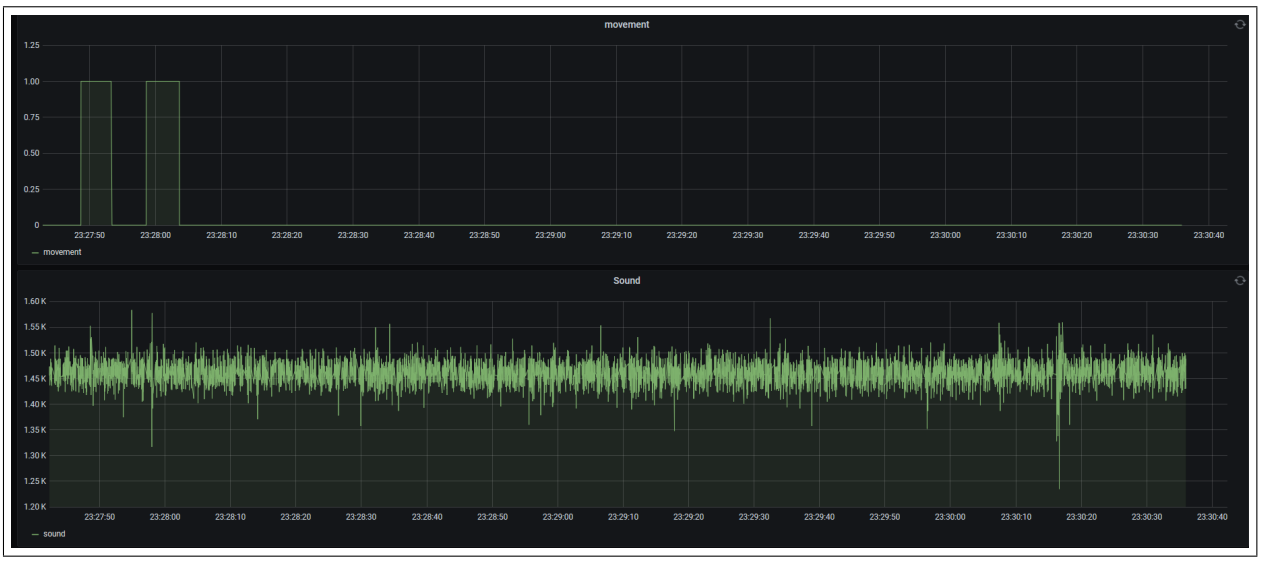

Figura 4.13: Dashboard de comportamento da criança.

Para fazer a coleta destes dados e mostrá-los na tela, o Grafana conecta-se ao banco de dados por meio da porta 3306 (padrão de conexões cliente-servidor no MySQL) utilizando uma credencial com as permissões necessárias para realizar consultas. Desta forma, o Grafana consegue fazer consultas diretamente às tabelas, sem necessidade de utilizar a API. A interface Web do Grafana fica então disponível na porta 3000 da máquina hospedeira, que foi mapeada na linha 36 do Apêndice B.1 para a porta 3000 do contêiner. Esta porta também teve de ser incluída nas exceções de *firewall* da máquina virtual na nuvem. Apesar deste serviço e contêiner não terem uma utilidade direta no funcionamento do nosso sistema, eles auxiliam muito no monitoramento dos sinais, já que os alertas apenas informam eventos. Este monitoramento foi essencial nas decisões dos algoritmos de detecção.

4.4.2 Módulo de Detecção e Notificação

O Módulo de Detecção e Notificação é responsável pela análise dos dados armazenados no banco de dados, pela detecção de eventos e pela notificação do usuário final. Dele fazem parte os contêineres que precisam que a Flask API e o banco de dados estejam em execução para que funcionem. São eles: *bot, cry, still* e *bad_env*.

Todos os contêineres deste módulo, com a exceção do contêiner *bot*, executam programas, escritos em Python pelos autores deste trabalho, que consultam dados do banco de dados via API, detectam eventos baseados nestes dados e então notificam os responsáveis pela criança por meio de um *bot* do aplicativo Telegram. Este procedimento é repetido indefinidamente (em *loop*) para que as detecções e notificações possam ser feitas em tempo real. Todos estes contêineres funcionam independentemente e em paralelo, isto é, o funcionamento de um contêiner não afeta a detecção e notificação realizadas por outro contêiner.

4.4.2.1 Contêiner bot

O *bot* do Telegram utilizado neste projeto nada mais é do que uma conta do aplicativo Telegram sem um número de telefone associado, que envia mensagens ou responde a comandos de acordo com a programação que lhe foi dada. Basicamente, ele utiliza a *Bot API* do Telegram usando uma biblioteca disponível para Python, que permite que ele realize várias ações dentro do aplicativo.

A criação do Bot é feita através de um outro Bot desenvolvido pela própria equipe Telegram chamado de *BotFather*, conforme mostra a Figura 4.14. Através de mensagens trocadas entre um usuário e ele, o *BotFather* cria uma conta para o novo Bot a partir de um nome e um nome de usuário escolhidos pelo usuário. O nome aparece no perfil da conta do Bot e o nome de usuário é utilizado em menções a ele durante a conversa. Depois de criado, o *BotFather* retorna uma senha que é usada na autenticação do novo Bot ao comunicar-se com a *Bot API*. Esta senha é extremamente importante, pois é ela que identifica o seu Bot nas chamadas de funções feitas dentro da aplicação. Qualquer pessoa com acesso a esta senha pode desenvolver uma aplicação que utiliza a conta do Telegram do seu Bot.

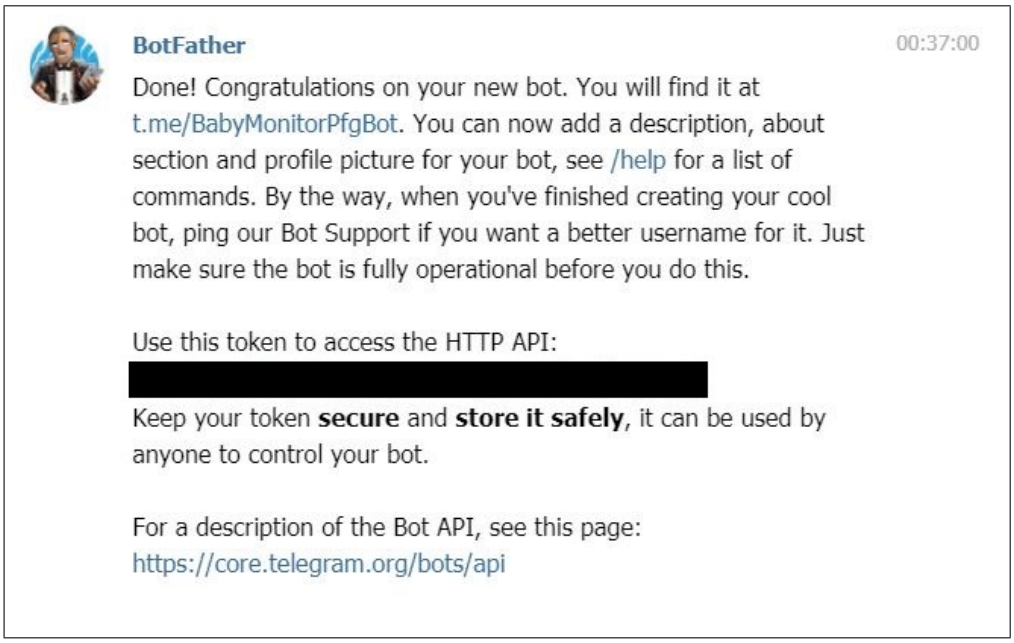

Figura 4.14: Criação do *Baby Monitor Bot* com o *BotFather*

O primeiro passo para a utilização deste Bot é incluí-lo em uma conversa em grupo. Iniciar uma conversa diretamente com o Bot também é possível, mas neste projeto, considera-se que todos os usuários do sistema tenham acesso a este grupo com ele para que todos sejam notificados dos eventos. Uma vez no grupo, o identificador de grupo/conversa deve ser verificado para que o Bot tenha uma referência de onde suas mensagens devem ser enviadas. Há algumas maneiras diferentes de se fazer isso, mas a que foi utilizada consiste em adicionar o Bot no grupo via

interface do aplicativo, mandar uma mensagem na conversa e então fazer uma consulta à API do Telegram passando a senha do Bot. A requisição HTTP GET enviada é respondida com uma série de informações sobre o Bot e, entre elas, estão as últimas mensagens recebidas e de onde vieram. Lá é possível identificar o *chat_id* da conversa em grupo.

Este contêiner utiliza a imagem *python:3.8.6-buster*, assim como os demais contêineres que executam programas desenvolvidos em Python. O código é bastante simples e faz uso principalmente da biblioteca *pyTelegramBotAPI*, além da biblioteca *requests* para que a API do contêiner app possa ser utilizada.

O código é apresentado no Apêndice B.2. O Bot é declarado passando como parâmetro a senha gerada pelo *BotFather*. Depois, funções são declaradas para que posteriormente sejam chamadas quando um usuário do grupo do aplicativo Telegram envia uma mensagem específica, como por exemplo "/start" ou "/help". O nosso Bot está programado para responder uma mensagem "/start" com a simples mensagem "Olá, eu sou o Bot monitor de bebê", conforme visto na Figura 4.15.

Figura 4.15: Resposta do Bot ao comando /start.

A funcionalidade mais interessante deste serviço é a resposta aos comandos "`/environment`" ou "`/env`", que contém as informações atuais de temperatura, umidade e luminosidade do quarto do bebê, como observado na Figura 4.16. A API é consultada em seus terminais `/temperature/now`, `/humidity/now` e `/light/now` e os valores retornados são formatados e enviados na conversa em grupo.

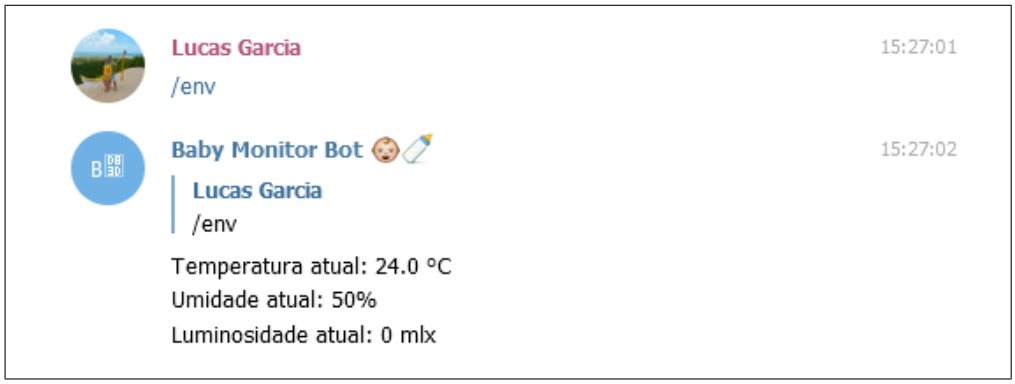

Figura 4.16: Resposta do Bot ao comando /env.

Por fim, é chamada a função `bot.polling()`, que inicia o Bot para que fique "escutando" a conversa até que um dos comandos seja executado por um participante do grupo. Esta função só pode ser utilizada uma vez entre quaisquer aplicações que utilizam este Bot (sob a mesma senha).

É importante frisar novamente que todo os contêineres executam seus programas de forma independente, em paralelo. Sendo assim, um contêiner pode ser finalizado ou iniciado sem afetar o funcionamento do resto do sistema. O programa executado no contêiner bot fornece informações do ambiente quando um usuário realiza uma requisição por meio de comando no aplicativo Telegram, mas isto é feito de forma independente das detecções de condições fora do ideal realizadas pelo programa executado no contêiner bad_env.

4.4.2.2 Contêiner cry

O programa executado no contêiner *cry* é responsável pela detecção e notificação de eventos de gemido, grito e choro. Estes eventos foram selecionados porque são os eventos sonoros mais recorrentes. Assim como os demais contêineres de detecção, ele executa um programa escrito em Python pelos autores deste trabalho, que consulta o banco de dados e avalia se houve um evento detectável ou não.

Primeiro, o programa consulta a Flask API do contêiner app por meio do terminal `/sound`, fazendo uma requisição HTTP GET. A API responde com os últimos 100 registros da tabela *sound*, lembrando que o espaçamento temporal entre estes registros é de 20 ms, totalizando 2 segundos de informação. Com os dados armazenados em um vetor, procura-se valores que indiquem um evento anormal, e se houver, faz-se uma segunda requisição HTTP GET à API, mas dessa vez pelo terminal `/movement` para obter informações de movimentação do bebê. A API responde com os últimos 100 registros da tabela *movement*, também representando 2 segundos de informação. Se valores que indicam movimentação do bebê são encontrados, a detecção do evento é realizada de acordo com os parâmetros explicados a seguir. Todo este procedimento é repetido a cada 2 segundos, de forma a monitorar a criança continuamente.

Para definir o algoritmo de detecção de eventos, foi primeiro observado o comportamento dos sinais de áudio e movimento do bebê. O primeiro aspecto que notamos foi a coincidência de detecções de movimentação e picos de áudio, como ilustrado na Figura 4.17.

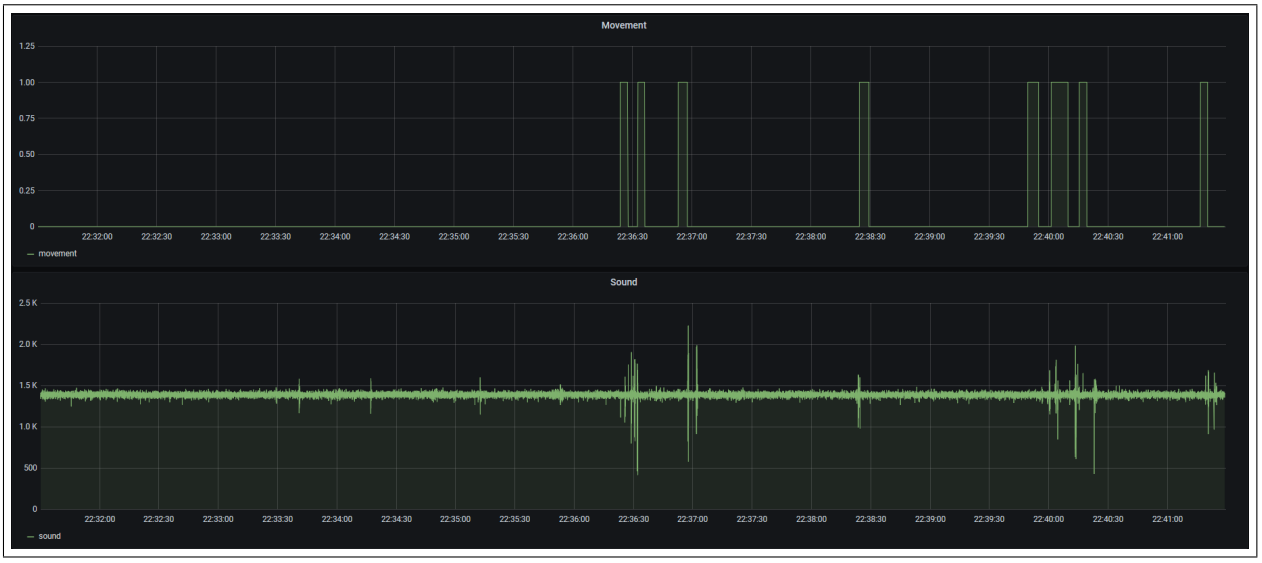

Figura 4.17: Monitoramento dos sinais de áudio (abaixo) e movimento (acima) do bebê.

A Figura 4.17 apresenta os valores de nível de ruído no ambiente, junto aos valores de detecção de movimento pelo sensor infravermelho, cobrindo um período de 10 minutos. Observe que toda vez que o bebê emite um som (neste caso gemidos), ele se movimenta quase que em sincronia com o ruído. Após um pouco mais de 1 hora de observação, tornou-se evidente que este comportamento se manteve consistente por todo o tempo observado. Em nenhum momento sequer o bebê emitiu um som sem se movimentar. Percebido isso, decidiu-se utilizar esta informação na detecção de gemidos, gritos e choro do bebê, uma vez que a análise cruzada de som e movimento dentro do carrinho anula vários falsos positivos que poderiam ocorrer devido a ruídos externos. Deve-se manter em mente que esta relação entre movimentação e emissão de ruídos foi identificada no comportamento do bebê observado especificamente e pode não ser válida para outras crianças.

O sensor de áudio, quando em um ambiente silencioso, registra valores que oscilam entre 1490 mV e 1440 mV, aproximadamente, com um valor médio aproximado de 1470 mV. A detecção é feita a partir de limiares. Os picos no sinal de áudio são identificados pelo sistema quando estão acima ou abaixo de limiares definidos ao redor do valor médio do sinal. Além dos valores muito acima do valor médio, os valores muito abaixo do valor médio também são indicativos de perturbação sonora devido à forma como a membrana do sensor se deforma. A pressão do ar causa uma vibração na membrana que faz com que, devido à sua elasticidade, gere valores de tensão acima e abaixo do valor médio encontrado durante o repouso.

Foram observados 25 eventos, entre gemidos e gritos, e constatou-se que em todos eles haviam picos com valores acima de 1600 mV ou abaixo de 1200 mV, portanto realizou-se uma segunda análise destes eventos para que fosse definida uma quantidade de picos necessários para configurar um ruído emitido pelo bebê, eliminando ainda mais falsos positivos. A análise retornou os resultados apresentados na Tabela 4.1.

Parâmetro	Valor Calculado
Média Aritmética	7,32
Variância	5,90
Desvio Padrão	2,43

Tabela 4.1: Parâmetros referentes às quantidades de picos encontrados nos 25 eventos observados.

Sendo assim, definiu-se que ao detectar-se 7 ou mais valores acima do limiar de 1.600 mV ou abaixo do limiar de 1.200 mV dentro de um período de 2 segundos, checa-se o sinal de movimento do bebê e se houver movimento, conclui-se que o bebê emitiu um som. Um exemplo de evento de gemido pode ser observado na Figura 4.18, onde 14 pontos encontram-se fora dos limiares de detecção.

Figura 4.18: Evento de gemido.

Além dos limiares já mencionados, foram configurados novos limiares para a detecção de picos mais altos que caracterizam gritos. Estes novos limiares foram definidos em 2.000 mV e 800 mV, necessitando de apenas 5 picos detectados no período de 2 segundos para a interpretação do ruído como um grito. A detecção de gritos acontece após a detecção de gemidos. Se entre os picos encontrados houver 5 ou mais pontos com valores acima de 2.000 mV ou abaixo de 800 mV, alerta-se um grito. Um exemplo de evento de grito é ilustrado na Figura 4.19, onde 6 pontos se encontram fora dos limiares de detecção de gritos.

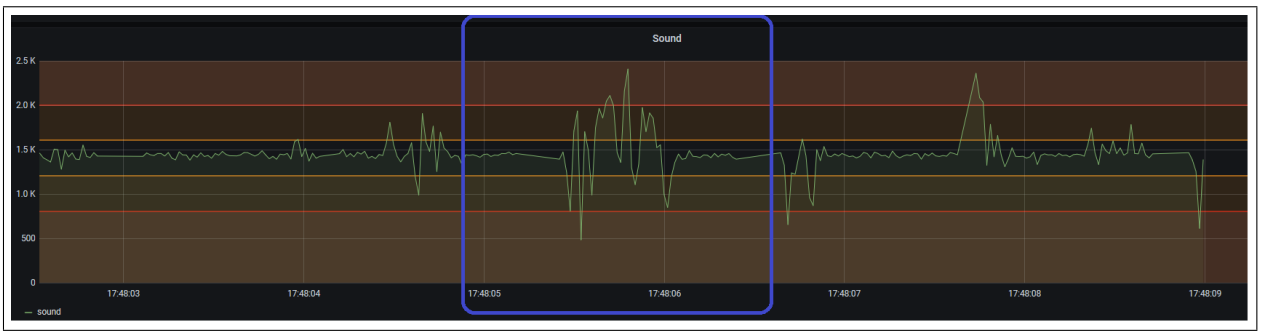

Figura 4.19: Evento de grito.

O choro é detectado quando se tem 30 ou mais picos fora dos limiares de 1.600 e 1.200 mV, independente dos valores encontrados, ou seja, ignora-se os limiares mais extremos. Isso foi definido devido à natureza mais contínua dos ruídos gerados pelo choro da criança. Um exemplo é apresentado na Figura 4.20, onde mais de 30 pontos são observados fora dos limiares de detecção de ruído.

Figura 4.20: Evento de choro.

Ao final da detecção de gemido ou grito, a execução do programa é interrompida por 7 segundos para reduzir a quantidade de alertas gerados. No caso de choro, esta interrupção dura 22 segundos. Todo o processo de detecção pode ser melhor compreendido por meio do fluxograma apresentado na Figura 4.21.

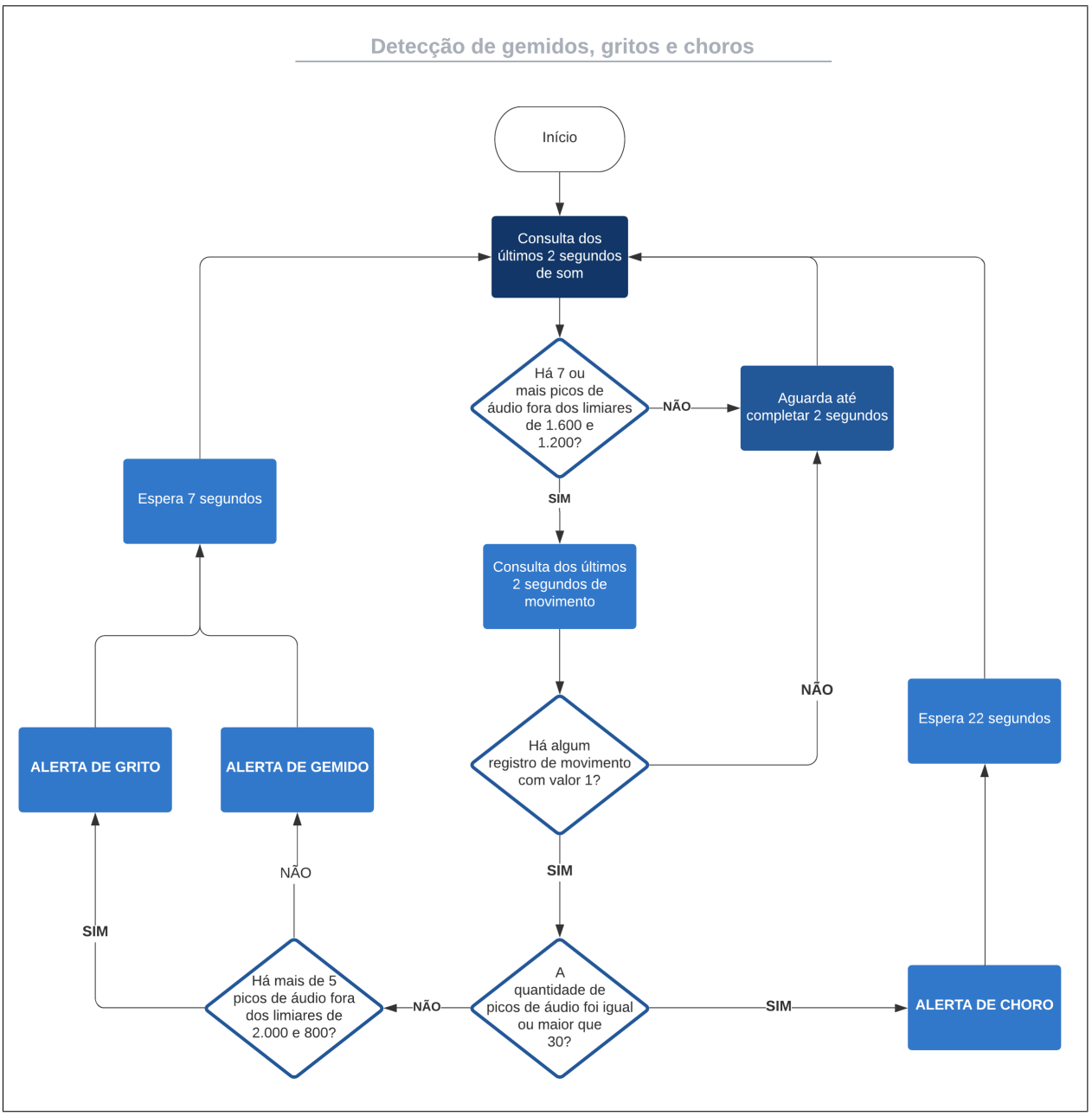

Figura 4.21: Fluxograma de detecção de gemidos, gritos e choros.

Além do arquivo que contém o código responsável pela detecção de eventos, há um arquivo importado que contém a declaração do Bot utilizando a mesma senha do contêiner *bot*, além das funções de alerta de gemido, grito e choro. Estas funções chamam a função `send_message` da biblioteca pyTelegramBotAPI passando o texto de alerta, horário da detecção e o *chat_id* que identifica a conversa em grupo em que os responsáveis estão. Todos os contêineres de detecção (cry, still e bad_env) importam um arquivo equivalente a este, com as devidas funções de alertas. Portanto, quando um evento sonoro é detectado, os pais recebem a mensagem no grupo do Telegram, conforme visto nas Figuras 4.22 e 4.23.

Figura 4.22: Alerta do Telegram para gemido e grito.

Figura 4.23: Alerta do Telegram para gemido, grito e choro.

4.4.2.3 Contêiner still

O programa executado no contêiner *still* tem o objetivo de identificar eventos de movimentação anormal, seja agitação excessiva ou ausência de movimentação. A partir da observação do bebê, percebeu-se que, mesmo em sono profundo, ele se movimenta periodicamente. Assim surgiu a ideia de alertar no caso do bebê estar sem se movimentar há mais de 10 minutos. Este alerta é importante também para identificar se o bebê deixou ou foi retirado do carrinho.

O alerta da agitação excessiva surgiu após um episódio em que uma pequeno pano caiu sobre o bebê e ele, confuso, se movimentou até que o objeto fosse removido. Como ele não emitiu nenhum som, a ideia de alertar a movimentação excessiva por períodos mais longos que 10 segundos surgiu. O programa que foi desenvolvido para realizar a detecção desses dois alertas e o envio das devidas notificações foi escrito pelos autores deste trabalho, utilizando a linguagem Python.

O programa consulta o banco de dados via APÌ no terminal `/movement/still` para acessar

os últimos 500 registros da tabela *movement*, que correspondem a 10 segundos já que o microcontrolador ESP32 envia 50 registros de movimento a cada segundo (sendo um dado coletado do sensor a cada 20 ms). A cada 10 segundos, o programa é executado novamente, fazendo uma nova consulta no banco de dados e executando novamente o algoritmo de detecção de eventos sobre estes novos dados, tornando possível assim o monitoramento contínuo da criança. Como o sensor de movimento PIR (consultar Seção 3.1.2.3) gera uma saída digital, isto é, indica apenas se há ou não há movimento sem especificar intensidade, os registros armazenados no banco de dados são binários, sendo 1 indicativo de que há movimento e 0 indicativo de que não há movimento. O programa utiliza contadores para acompanhar a quantidade de 1's e 0's seguidos, que representam tempo de agitação e tempo de ausência de movimento, respectivamente. Se 10 minutos de inatividade contínuos são percebidos, o programa envia aos usuários o alerta inicial de inatividade. Se os contadores chegam a 20 minutos de inatividade, um segundo alerta de é enviado indicando que a inatividade se estende há 20 minutos. Estes dois alertas de inatividade podem ser observados na Figura 4.24. Há também o alerta de agitação excessiva que é enviado quando é detectada movimentação contínua por períodos de 10 segundos ou mais.

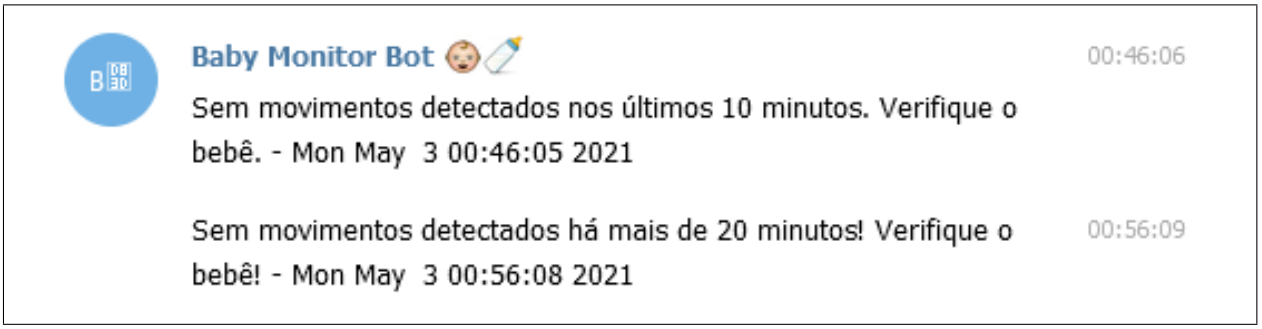

Figura 4.24: Alertas de inatividade.

4.4.2.4 Contêiner bad_env

O programa executado no contêiner *bad_env* é responsável pelas notificações de ambiente impróprio. Quando as condições de temperatura, umidade do ar e luminosidade (durante a noite) não estão adequadas para o bebê, os usuários são avisados para que uma ação seja tomada.

Segundo Higuera [59] e a *Lullaby Trust* [60], uma organização inglesa dedicada à disseminação de informações e ao combate à Síndrome da Morte Súbita do Lactente (SMSL), a temperatura mais confortável para o bebê dormir é entre 16 e 20 ºC, além de ser mais seguro para evitar a SMSL. Considerando as temperaturas tropicais do Brasil e a utilização de menos camadas de roupas nos bebês, os alertas foram configurados para acionarem os usuários quando as temperaturas estiverem abaixo de 16 ºC ou acima de 26 ºC. O sensor de temperatura precisa de um pouco de atenção porque, ao ligar o kit XDK, a placa esquenta e as leituras do sensor vão aumentando. Eventualmente ele estabiliza em uma temperatura alguns graus acima da temperatura ambiente.

Utilizando um termômetro externo, foi identificada a diferença entre a temperatura ambiente e a temperatura marcada pelo sensor para que um *offset* negativo fosse utilizado para compensar o calor da placa. Este valor foi definido em -5500 mºC. Desta forma, quando a temperatura marcada pelo sensor estabiliza, os valores coletados se tornam muito semelhantes à temperatura identificada pelo termômetro externo.

A *Environmental Protection Agency* (EPA) dos Estados Unidos [61] define que a umidade relativa do ar ideal para ambientes internos é entre 30% e 60%. Esse valor considera baixa umidade que pode prejudicar o sistema respiratório das pessoas e também alta umidade que pode facilitar o desenvolvimento de mofo no ambiente. Como estes valores se referem a crianças e adultos, mas não especificamente a bebês, que são mais suscetíveis a más condições, o alerta foi configurado para ser disparado quando a umidade relativa atinge 40% ou menos. Um alerta para alta umidade não foi considerado importante para o sistema, uma vez que o controle de mofo foge do escopo do projeto.

Por fim, foi implementada também uma notificação para o caso em que uma cortina tenha sido deixada aberta e o dia começou a amanhecer ou então alguma lâmpada foi esquecida ligada. O sensor de luminosidade indica 0 lúx quando o quarto encontra-se confortavelmente escuro. Quando a luz do teto é acesa, o valor salta para 14.400 mlx. Ficou definido então que o alerta seria disparado quando a iluminância atingisse 10.000 mlx (ou 10 lx).

Para fazer estas detecções, o sistema primeiro envia requisições HTTP GET à API pelos terminais `/temperature`, `/humidity` e `/light`. Os 3 terminais respondem com os últimos 20 registros de suas respectivas tabelas, correspondendo a 20 segundos de leitura uma vez que a Bosch XDK envia um valor a cada segundo. Estes valores são armazenados em vetores para análise. A consulta ao banco de dados via API e o algoritmo de detecção são executados a cada 20 segundos, de forma que o ambiente possa ser monitorado continuamente.

Primeiro avalia-se a temperatura. O vetor de dados de temperatura é percorrido em busca de valores abaixo do limiar de 16 ºC ou acima do limiar de 26 ºC. Se 20 registros (que correspondem a 20 segundos de coleta, visto que o kit XDK registra valores uma vez por segundo) apresentam valores acima do limiar superior, o alerta de alta temperatura é enviado aos usuários. De forma análoga, se 20 registros são encontrados abaixo do limiar inferior, o alerta de baixa temperatura é enviado aos usuários. A contagem de registros é mantida entre execuções do programa, portanto é possível atingir o vigésimo registro de alta ou baixa temperatura sem precisar percorrer todo o vetor de temperaturas. Um exemplo de alerta de alta temperatura é apresentado na Figura 4.25.

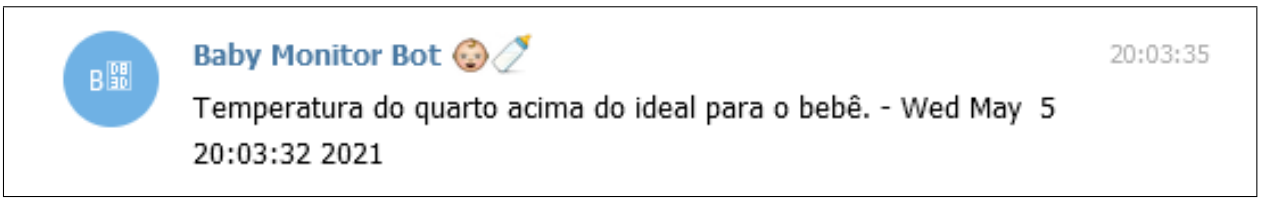

Figura 4.25: Alerta de alta temperatura.

Depois avalia-se a umidade. Utilizando um contador, registra-se a quantidade de valores consecutivos abaixo do limiar de 40%. Se um valor acima deste limiar é encontrado, o contador é zerado. Se não, o contador é mantido até que atinja 20, quando um alerta é enviado. Quando o vetor de dados de umidade (que contém 20 registros) é percorrido até o fim, o valor do contador é mantido e utilizado na análise dos próximos 20 segundos. O alerta de baixa umidade indica que a umidade encontra-se abaixo do ideal há mais de 20 segundos, conforme exemplificado na Figura 4.26.

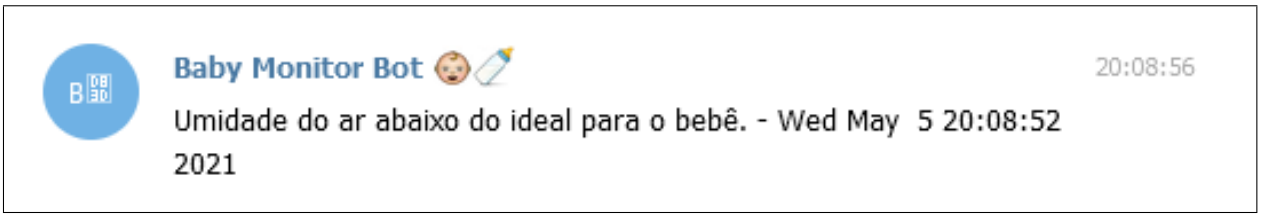

Figura 4.26: Alerta de baixa umidade.

Por fim avalia-se a luminosidade. De forma similar, um contador registra a quantidade de valores consecutivos acima do limiar de 10.000 mlx. Quando o contador atinge 10, o alerta de luminosidade é disparado. Um exemplo de alerta de luminosidade é apresentado na Figura 4.27.

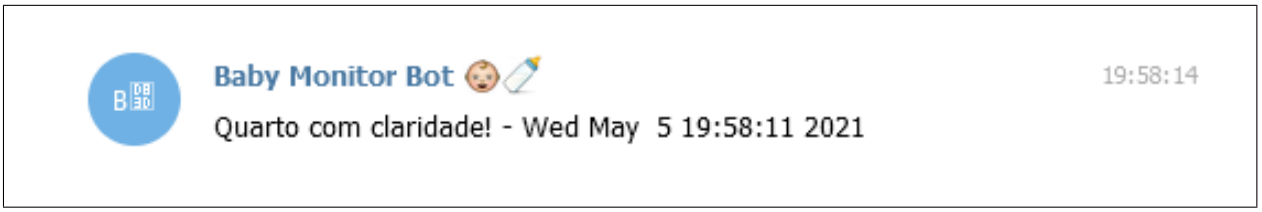

Figura 4.27: Alerta de alta luminosidade.

Com a avaliação do ambiente finalizada, se um ou mais alertas foram gerados, o programa é interrompido por 5 minutos a fim de reduzir a quantidade de notificações enviadas. Se nenhum alerta foi gerado, o programa aguarda o fim dos 20 segundos para fazer uma nova requisição à API e analisar o ambiente novamente.

Capítulo 5

Resultados

5.1 Visão Geral

Com o objetivo de realizar testes de eficácia do sistema, o ambiente foi primeiramente preparado posicionando-se no carrinho os sensores e dispositivos de sensoriamento de forma que captassem corretamente os dados do ambiente em que o bebê se encontra e os dados referentes ao comportamento do próprio bebê (som e movimento). Feito isso, todos os componentes do sistema são ligados para que a coleta e envio dos dados, bem como os programas de detecção executados na nuvem, sejam iniciados. Uma vez que o sistema está em pleno funcionamento, pode-se avaliar a consistência dos alertas quando comparados aos eventos observados presencialmente. O objetivo dos testes, com observações presenciais, é avaliar a acurácia do envio de alertas por meio da comparação de alertas de eventos detectados corretamente, alertas detectados incorretamente, e eventos observados que não foram alertados pelo sistema.

Os alertas foram validados com observações em ambiente controlado, de modo que as capturas pudessem ser feitas no período da noite, quando chegava a hora de dormir do bebê. Ao acionar o sistema e colocar o bebê no carrinho, começam as observações de cada som ou movimento, fazendo-se registros. Estes registros, posteriormente, são comparados com os alertas que o *bot* do Telegram, mencionado na Seção 4, envia. Dessa forma pode-se analisar a eficiência do sistema de alertas. A coleta também foi feita em um cenário sem o bebê no carrinho e com ruído externo de intensidade alta, produzido por uma pessoa ao telefone. Neste contexto não houve alertas falsos (o único alerta enviado foi de ausência de movimento, pois não havia bebê no carrinho).

Para este projeto, os resultados obtidos foram colhidos especificamente de uma criança com idade de dois meses. Isso se deu pelo fato de que o projeto foi concluído para a etapa de captura das amostras em meio à pandemia de COVID 19. Com isso, outras crianças não puderam ser visitadas e assim terem seus dados coletados. Desse modo, não pôde-se observar as diversas situações que fazem os bebês chorarem e ficarem agitados. Outra consideração a ser feita, como

mencionado na Seção 1.5, é que durante o período de observações a criança observada ainda estava dormindo no quarto dos pais, ao lado da cama, razão pela qual os testes foram realizados no carrinho adaptado para moisés (cama semelhante a uma cestinha, mini-berço ou berço portátil especificamente projetada para bebês desde o nascimento até cerca de quatro meses).

Os dados considerados para este projeto foram capturados na semana do dia 27 de abril de 2021 ao dia 03 de maio de 2021, observando-se o bebê por 1 hora em cada dia, totalizando 7 horas. Antes deste período de testes oficiais, outros testes foram conduzidos com o intuito de aperfeiçoar os algoritmos de detecção e o posicionamento dos sensores. Durante este período de testes provisórios, algumas funcionalidades foram abandonadas, como por exemplo a detecção de apneia, que apresentou desempenho ruim com baixas taxas de acurácia.

5.1.1 Montagem do Protótipo

Nesta Seção será apresentada a montagem dos componentes de *hardware*, posicionamento de sensores e a instalação do sistema no carrinho. Como mencionado na Seção 3.1.2 o microcontrolador ESP32 vem com 30 pinos com várias funções. Neste projeto utilizou-se os pinos *GPIO* 16 e 35 para receber os sinais de movimento e áudio. O pino 16 recebe a saída digital do sensor de movimento e o pino 35 recebe a saída analógica do sensor de áudio. Na Figura 5.1, pode-se observar como a montagem dos sensores com o microcontrolador foi feita. Observa-se os sensores de áudio MAX9418 e de movimento PIR ligados ao microcontrolador, que faz a função de processamento dos dados e fornece alimentação de 3.3 V aos sensores. Para a ligação entre os componentes de *hardware*, utilizou-se uma matriz de contato com 400 pontos adaptada ao microcontrolador ESP32.

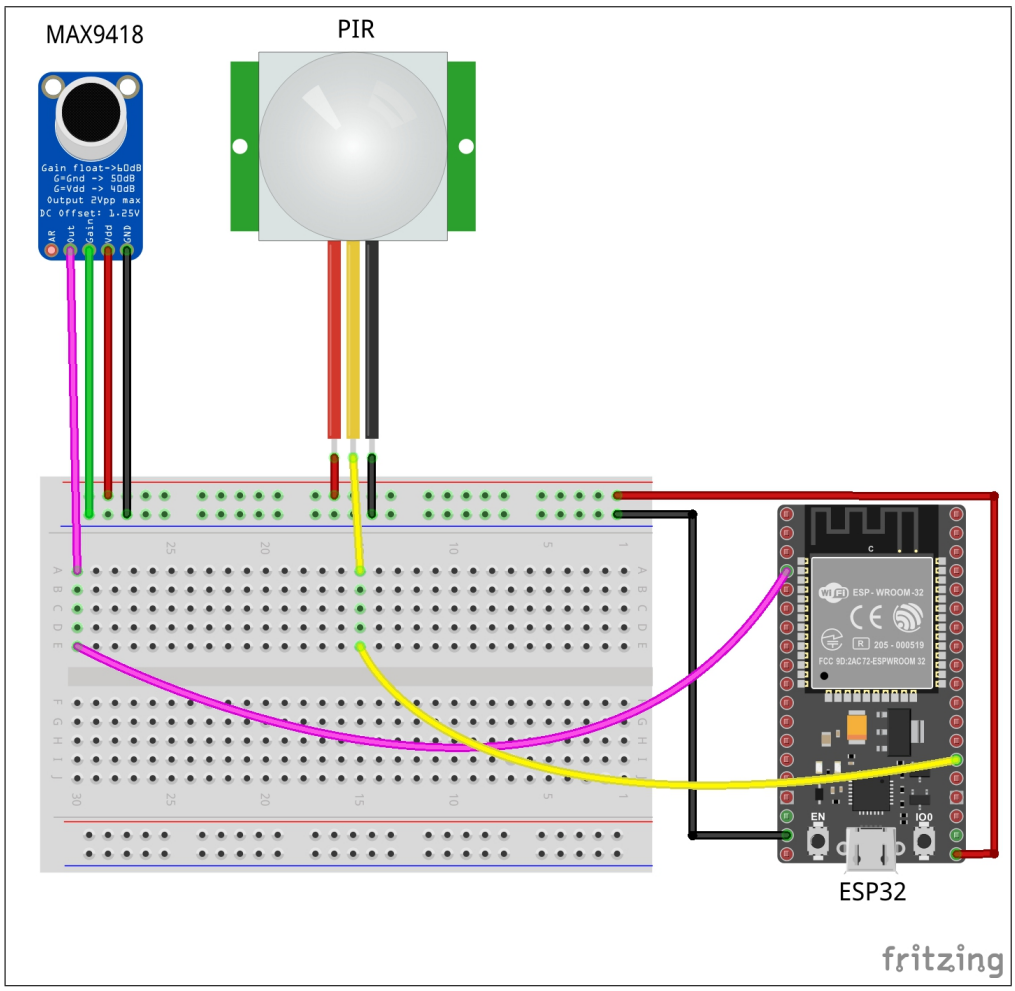

Figura 5.1: Esquema de montagem do sensor de movimento (PIR), sensor de áudio (MAX9418) e o microcontrolador ESP32, utilizando fios para a ligação dos pinos GPIO 16 e 35 aos terminais dos sensores.

No protótipo construído para esse projeto, o microcontrolador ESP32, o kit da Bosch e os sensores estão colocados no carrinho de modo que não forneça nenhum risco à integridade física da criança, que encontra-se a uma distancia de 20 centímetros para o sensor de áudio e 15 centímetros para o sensor de movimento e para o kit da Bosch. Segundo os manuais dos sensores de áudio e movimento, ambos capturam informações em distâncias maiores que 1 metro, portanto não houve problemas com o distanciamento dos sensores, afinal ficaram próximos ao bebê. Na Figura 5.2, pode-se observar o resultado da montagem dos componentes de *hardware*, a matriz de contato adaptada e as ligações entre os sensores e o microcontrolador ESP32. Na Figura 5.3 e na Figura 5.4 pode-se ver o resultado final da instalação do sistema de detecção no carrinho de bebê. Os sensores foram fixados em lados opostos do carrinho. O microcontrolador ESP32 foi colocado na parte de trás, e a XDK foi posicionada na parte da frente, de modo que possa captar melhor os dados de ambiente, principalmente no que diz respeito a iluminância. Com essa disposição física das placas o sistema não apresenta risco para o bebê e permite fácil acesso à bateira, que alimenta

ambas as placas por meio de cabos USB. Essa alimentação é feita por uma bateria externa recarregável que também fica acoplada no carrinho. No protótipo, a bateria é fixada no compartimento inferior do carrinho para, novamente, não trazer riscos à integridade física da bebê.

Figura 5.2: Montagem do *hardware*.

Figura 5.3: Montagem do carrinho de bebê — visão de dentro.

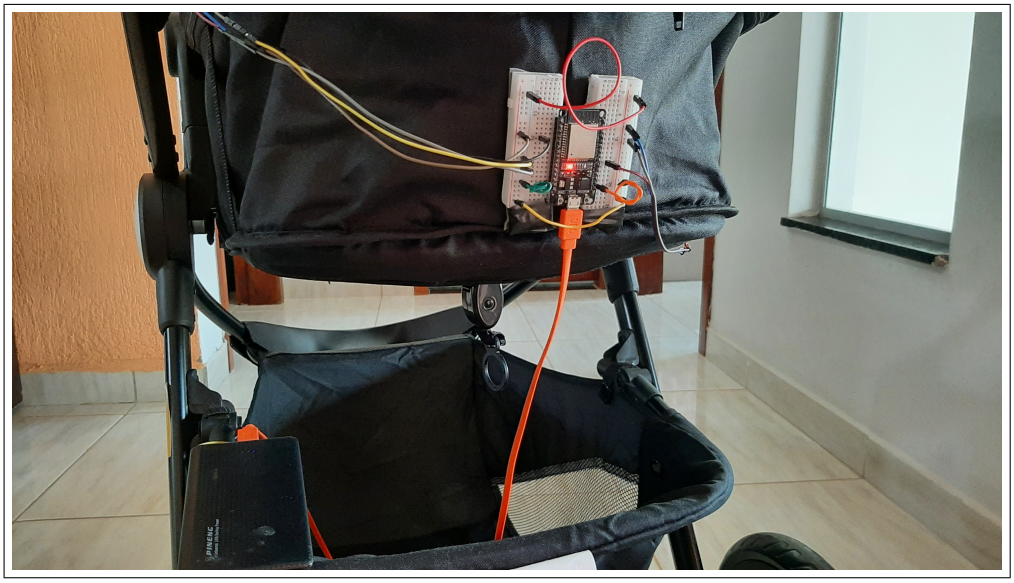

Figura 5.4: Montagem do carrinho de bebê — parte de trás.

5.1.2 Coleta de Dados

Os testes foram conduzidos durante uma semana, observando-se o bebê por uma hora em cada dia. Quando o bebê produz algum som, um registro de interpretação do som (gemido, grito ou choro) era anotado pelo observador presente, assim como o horário do evento. Enquanto isso, o grupo do Telegram foi monitorado para acompanhamento das notificações: envio (ou não) e a natureza das notificações. Assim, é possível fazer a análise cruzada das informações anotadas pelo observador presente com as notificações recebidas no aplicativo Telegram.

As observações foram feitas sempre à noite, quando a criança era colocada para dormir. Em alguns dias, ela se apresentava mais agitada que em outros, devido ao disgnóstico mencionado na Seção 5.1 que fica visível nos dados coletados. Cada evento de gemido, grito e choro é registrado pelo observador presente no momento em que acontece e o envio e natureza do alerta são então verificados no grupo do Telegram.

A Tabela 5.1 apresenta as informações coletadas referentes às detecções de gemidos durante os 7 dias de observação (por períodos de 1 hora em cada dia). Apesar dos gemidos acontecerem durante o sono do bebê, eles podem ser um indicativo de que a criança está desperta quando ocorrem seguidamente. Na Tabela 5.1, são considerados:

- **Gemidos Observados:** gemidos observados presencialmente. Gemidos são curtos e de baixa intensidade, mas a interpretação cabe sempre ao observador, que foi o mesmo durante todos os testes;
- **Gemidos Detectados Corretamente:** são os gemidos que foram observados e detectados corretamente. São "verdadeiros positivos";
- **Falsos Negativos:** são os gemidos que foram observados, mas não foram alertados. Eventos que o sistema não detectou corretamente;
- **Falsos Positivos:** são os alertas que foram enviados, mas não observados. Também inclui as detecções erradas como quando um grito é detectado como gemido, por exemplo;
- **Total de Eventos:** inclui todos os gemidos detectados corretamente, todos os falsos positivos e todos os gemidos não detectados. Representa o universo de eventos para que se consiga chegar em uma taxa de acerto;

Dia	Gemidos Observados	Gemidos Detectados Corretamente	Falsos Negativos	Falsos Positivos	Total de Eventos
27/abr	30	27	3	0	30
28/abr	21	19	2	0	21
29/abr	38	34	4	2	40
30/abr	25	24	1	0	25
01/mai	65	59	6	1	66
02/mai	20	18	2	1	21
03/mai	47	42	5	2	49
TOTAL:	246	223	23	6	252

Tabela 5.1: Registros de detecção de gemidos.

A partir destes dados, uma taxa de acerto é calculada considerando a quantidade de gemidos detectados corretamente (verdadeiros positivos) dentre todos os eventos, ou seja, a taxa representa uma proporção de acertos para erros (falsos positivos e falsos negativos). Com uma taxa de acertos de 88,5%, considera-se que o sistema apresentou bons resultados na detecção de gemidos. Como trata-se de um evento bastante comum, é mais fácil obter uma quantidade satisfatória de ocorrências para se avaliar o desempenho do sistema.

Os gemidos não detectados foram na maioria gemidos muito baixos, que não foram detectados propriamente pelo sensor acústico, e gemidos durante os quais o bebê não se moveu. Os poucos falsos positivos foram em sua maioria gritos interpretados como gemidos.

A Tabela 5.2 apresenta os dados referentes a eventos de gritos coletados durante os mesmos períodos de observação que os gemidos. Os gritos representam um evento mais pontual, de criticidade maior, que normalmente indica que a criança está desconfortável. Diferente dos gemidos, que são bastante comuns mesmo quando a criança está confortável, os gritos de crianças pequenas (menores de 1 ano de idade) normalmente indicam necessidade de atenção.

Dia	Gritos Observados	Gritos Detectados Corretamente	Falsos Negativos	Falsos Positivos	Total de Eventos
27/abr	1	1	0	2	3
28/abr	0	0	0	0	0
29/abr	1	1	0	1	2
30/abr	0	0	0	0	0
01/mai	3	2	1	0	2
02/mai	1	1	0	1	2
03/mai	2	2	0	0	2
TOTAL:	8	7	1	4	12

Tabela 5.2: Registros de detecção de gritos.

As colunas da Tabela 5.2 acima têm os mesmos significados que as colunas da Tabela 5.1: eventos observados, eventos detectados corretamente, eventos não detectados, falsos positivos e total de eventos. A taxa de acerto encontrada para eventos de grito foi de 58,3%.

Gritos e choros são muito menos comuns que gemidos. Portanto a quantidade de eventos de gritos capturados foi bastante reduzida. O grito foi o evento com a maior taxa de falsos positivos devido a outros eventos como tosses e espirros, que geram picos de altas intensidades no sinal de áudio. A baixa taxa de acertos pode ser justificada pela escassez de eventos reais de grito, o que faz com que os falsos positivos "se sobressaiam" nas estatísticas. Note, que apesar dos falsos positivos, a taxa de gritos não detectados foi bastante baixa.

Choros também não aconteceram muitas vezes durante as 7 horas de observação. Em alguns dias, nenhum evento de choro foi observado ou detectado. Os falsos positivos são compostos em sua totalidade por gemidos ou gritos prolongados erroneamente identificados como choro. A Tabela 5.3 apresenta os dados relativos a eventos de choro.

Dia	Choros Observados	Choros Detectados Corretamente	Falsos Negativos	Falsos Positivos	Total de Eventos
27/abr	2	2	0	0	2
28/abr	2	2	0	0	2
29/abr	0	0	0	1	1
30/abr	0	0	0	0	0
01/mai	2	2	0	2	4
02/mai	1	1	0	0	1
03/mai	1	0	1	0	1
TOTAL:	8	7	1	3	11

Tabela 5.3: Registros de detecção de choros.

Como pode-se observar na Tabela 5.3, a detecção de choro se comportou de forma similar à detecção de gritos, principalmente devido à escassez destes eventos. Note que assim como na detecção de gritos, a detecção de choro também teve uma baixíssima taxa de choros não detectados. A taxa de acerto no caso de eventos de choro foi idêntica à taxa encontrada para gritos: 63,6%.

As detecções de ausência de movimento e movimento excessivo não falharam nenhuma vez, com a exceção dos momentos em que o bebê foi retirado do carrinho para ser amamentado ou ter sua fralda trocada, o que gerava um alerta de ausência de movimento. Outra particularidade observada foi que o alerta de movimentação excessiva (que é gerado quando o bebê apresenta movimentação contínua por mais de 10 segundos, independentemente de dados de áudio) aconteceu frequentemente em meio aos gemidos em sequência, ou pouco antes de um choro, o que mostra que o sistema pode, de certa forma, auxiliar os pais a evitar o estresse do bebê verificando o que está gerando a agitação antes do choro acontecer.

Os alertas de ausência de movimento por 10 minutos foram pouco recorrentes e os alertas de 20 minutos não foram disparados nenhuma vez. Esses alertas de ausência de movimento, quando ocorreram, aconteceram porque o bebê estava respirando levemente e em sono profundo. Como de fato não há movimento além da expansão e contração da caixa torácica do bebê, considera-se que este programa de detecção de eventos de movimento não apresentou falhas. De qualquer forma, como dentre todos os testes oficiais e extra-oficiais feitos, o bebê se manteve imóvel por mais de 10 minutos em pouquíssimas ocasiões, considera-se que, pelo menos para o bebê observado neste projeto, a ausência de movimentos além da respiração por períodos mais longos que 10 minutos são indicativos de que algo pode estar errado com a criança.

Os alertas ambientais também não apresentaram erros. Como a detecção é baseada em limiares simples, as alterações de temperatura e iluminância foram captadas conforme esperado. O alerta de baixa umidade não pôde ser testado, uma vez que não foi possível reduzir a umidade do ar por intervenção humana. Já o alerta de luminosidade é disparado toda vez que a luz é acesa no quarto, conforme programado.

Como ao ligar o kit XDK, a placa esquenta e as leituras do sensor vão aumentando, foi utilizado um *offset* negativo para compensar o calor da placa. Este valor foi definido em -5500 mºC. Devido a este comportamento, foi muito comum um alerta de baixa temperatura ser enviado ao ligar o kit XDK, mas com o tempo a temperatura no interior do equipamento aumenta até que os valores registrados pelo sensor se aproximam dos valores reais (se comparados com a temperatura medida com termômetros externos) e assim o sistema estabiliza. Como o programa de detecção de condições ambientais impróprias é interrompido por 5 minutos após uma detecção, este falso alerta é enviado apenas uma vez, visto que em 5 minutos a temperatura no interior do kit já estabilizou.

Capítulo 6

Conclusões

O projeto apresentado neste documento teve como objetivo não só desenvolver um sistema de monitoramento de bebês, mas também implementar uma arquitetura IoT, que representa uma área em foco atualmente. É por isto que, diferentemente da maioria dos demais trabalhos citados, manteve-se a intenção de emprego das entidades que configuram um típico sistema IoT. Os pontos principais a serem considerados são o armazenamento e o processamento realizado na nuvem, que traz flexibilidade de acesso, abstração às necessidades de manutenção, isolamento de microsserviços, otimização de processos e redução de custos. Conseguiu-se integrar dois dispositivos diferentes, um *gateway* que centraliza e organiza o envio de mensagens e o processamento na nuvem, configurando um típico sistema IoT que pode ser expandido para incorporar mais dispositivos e sensores.

O kit XDK da Bosch provou-se bastante versátil em termos de funcionalidades. Por ter diversos sensores embarcados no kit, uma série de informações diferentes pode ser coletada do ambiente sem a necessidade de adquirir novos equipamentos. Além da vasta quantidade de sensores, o kit XDK também conta com módulos de comunicação via Wi-Fi e Bluetooth, que também facilitam bastante o desenvolvimento de aplicações IoT. O desenvolvimento de código pelo *XDK Workbench* se mostrou bem documentado, além de possuir uma comunidade considerável de pessoas interessadas no kit, o que torna a busca por informações mais simples. Dito isto, o dispositivo também apresentou algumas limitações quanto ao *hardware*. Estas limitações serão explicadas na Seção 6.1.

O microcontrolador ESP32, provou ser bastante prático aliando baixo consumo de energia, boa conectividade Wi-Fi e facilidade de programação. Uma característica importante da placa é a sua portabilidade, visto que tem tamanho reduzido e pode ser facilmente posicionada em qualquer lugar do carrinho. O módulo Wi-Fi e a capacidade de memória de 512 KB facilitaram a utilização do microcontrolador, mostrando que exerce bem funções de recebimento, envio e processamento de dados em aplicações IoT. O desenvolvimento de código fazendo uso da *Arduino IDE* melhora a eficiência de trabalho, pois possui bastante documentação na Internet. Por outro lado, o uso do

microcontrolador ESP32 exigiu que fossem feitas adaptações na matriz de contato devido à forma que os pinos são dispostos.

A utilização de aplicações distribuídas em contêineres possibilitou que, durante o desenvolvimento do projeto, os programas utilizados no sistema fossem migrados de máquina a máquina sem dificuldades. Durante boa parte do desenvolvimento do trabalho, o processamento foi feito localmente, já em contêineres. Quando os contêineres foram migrados para o serviço em nuvem, o processo de mudança exigiu apenas a instalação da aplicação Docker na nova máquina virtual. Além disso, durante os ajustes dos algoritmos de detecção, foi possível executar apenas um ou dois contêineres para fazer a depuração de erros e otimização do código, uma vez que cada contêiner operava de forma independente e assíncrona. A utilização dos serviços na nuvem da Google por meio da *Google Cloud Platform* foi mais simples do que configurar um virtualizador em uma máquina local. Com apenas alguns cliques, a máquina virtual é disponibilizada para acesso via SSH. Apesar do dimensionamento da máquina ter sido uma preocupação, o programa que fornece U$300,00 a novos usuários foi suficiente para manter a máquina virtual *online* e com bom desempenho por quase 3 meses.

Quanto à experiência do usuário, a utilidade prática do sistema foi confirmada pelos responsáveis pelo bebê que participaram dos testes. Acredita-se que ao estarem atentos aos sinais oriundos do bebê pode-se evitar situações que possam causar acidentes que resultem em morte súbita infantil inesperada, posto isto, o sistema cumpriu bem sua função ao emitir alertas de acordo com as situações em que foi exposto, permitindo aos responsáveis suprir, com agilidade, as necessidades do bebê. Apesar de utilizarem uma babá eletrônica convencional, houve momentos em que o sistema alertou eventos que eles não haviam percebido. Por exemplo, em um momento em que estavam na área do quintal da casa enquanto os vizinhos estavam com som alto, o choro do bebê proveniente da babá eletrônica não foi ouvido pelo casal, mas a notificação no telefone celular os alertou e assim foram verificá-la. Além disso, apesar da babá eletrônica ter a funcionalidade de transmissão contínua de vídeo, o usuário precisa ativamente acessar a função de vídeo para verificar o bebê no monitor. Com o sistema proposto, nenhum novo objeto precisa ser carregado pelos pais, apenas o seu *smartphone* que já é parte do cotidiano de todos.

6.1 Desafios

Ao longo do desenvolvimento do projeto, várias ideias foram abandonadas devido a empecilhos e limitações. O projeto começou quando nos foi fornecida o kit Bosch XDK. A disponibilidade do sensor acústico e o futuro nascimento do bebê observado nos inspirou a desenvolver um sistema que pudesse captar o sinal de áudio e processá-lo para que se pudesse determinar se houve choro ou não. O primeiro passo no desenvolvimento deste projeto seria a captação e armazenamento dos dados para que então fosse feito um estudo do espectro de frequência do sinal

de choro e assim fosse possível identificar padrões. Se algumas faixas específicas de frequência, provavelmente mais altas, fossem proeminentes, isso seria usado na decisão de detecção.

Quando começamos a programar o kit XDK para fazer a coleta e envio dos dados de áudio, percebemos algumas limitações. A primeira foi o fato de que não é possível amostrar os dados de áudio enquanto outros sensores estão habilitados. A documentação do fabricante Bosch [41] não fornece muitas informações sobre o motivo, mas nota explicitamente que não é possível utilizar o sensor acústico enquanto os demais estão habilitados. Dessa forma, se o sensor acústico fosse utilizado, seria impossível fazer uso dos sensores ambientais do kit, o que culminaria na necessidade de aquisição de mais sensores individuais para que a ESP32 fizesse o monitoramento das condições do ambiente.

Além disso, segundo o Teorema da amostragem de Nyquist–Shannon, para que obtivéssemos um espectro de frequência que mantivesse todas as características espestrais do sinal original, teríamos que amostrá-lo, no mínimo, a uma frequência duas vezes maior do que a maior frequência do sinal original. Como mencionado por Torres, Battaglino e Lepauloux [62], o choro de bebês alcança tipicamente frequências entre 250 Hz e 600 Hz, ou seja, para garantirmos um espectro fiel ao sinal original, teríamos que amostrar a uma taxa de 1,2 kHz.

Ao tentarmos implementar esta taxa de amostragem, descobrimos que, por uma limitação do FreeRTOS, os temporizadores não podem ser configurados para dispararem interrupções a frequências maiores que 1 kHz. Tentou-se então trabalhar com os temporizadores de *hardware*, que utilizam relógios de frequências de até 48 MHz (utilizando todas as marcações do processador). Mas, infelizmente, devido à forma como o sensor AKU340 é conectado ao microcontrolador, a amostragem em altas frequências não foi consistente o suficiente para ser usada. Passamos então a considerar a utilização do microcontrolador ESP32 para a coleta dos sinais de áudio e movimento, enquanto o kit Bosch XDK fez a coleta das informações do ambiente.

Foi considerado também o monitoramento da respiração do bebê por meio de sensores. A forma menos intrusiva de se fazer isso seria utilizando um sensor piezo-elétrico, cuja tensão de saída varia de acordo com a distância entre duas pastilhas. Quando pressionadas, elas se aproximam e a tensão aumenta. Sendo assim, considerou-se a possibilidade da expansão da caixa torácica da criança pressionar o sensor e assim obtermos um sinal de respiração.

Os primeiros testes foram feitos com um adulto e inicialmente não trouxeram bons resultados. Utilizando uma peça pontiaguda que concentrava a força em um ponto central do sensor, aumentando a pressão, o sinal de respiração do adulto se tornou bastante característico e utilizável. O teste com o bebê foi feito posicionando-se o sensor abaixo de um fino colchão sobre o qual ele estava deitado, mas os resultados não foram bons. A baixa massa do bebê e a pressão gerada pela expansão da caixa torácica não foram suficientes para apresentar grandes variações na tensão de saída do sensor. Infelizmente, esta ideia teve que ser abandonada.

6.2 Trabalhos Futuros

Como trabalhos futuros após o desenvolvimento desse projeto, pode-se incluir a realização de experimentos mais prolongados de observação, e, também a inclusão de testes em diferentes crianças de várias idades, até 12 meses. A presença de uma equipe médica de pediatria para validação e otimização dos limiares de detecção seria, também, uma agregação bastante adequada.

Assim como foi feito por Dubey e Damke [14], a utilização de redes neurais trariam ganhos significativos para a detecção de choro. O emprego de técnicas de aprendizado de máquina na identificação de sinais sonoros de choro possibilitariam a utilização do sistema em diversos ambientes e cenários diferentes. Com isso, a inteligência artificial conseguiria distinguir o choro do bebê em meio a vários outros tipos de som e também possibilitaria distinguir o motivo do choro.

Para que isso fosse possível, outro módulo de captura de áudio deveria ser empregado, uma vez que os sensores utilizados neste projeto são sensores de "ruído do ambiente", e não podem ser utilizados como microfones com os quais se pode reconstruir o áudio posteriormente à captura. Isso se deve ao fato destes sensores gravarem apenas o lado positivo do sinal, descartando informação valiosa para a sua reconstrução. Assim como foi feito por Lobo et al. [20], é possível utilizar o sensor de ruído do ambiente como ativador de um microfone USB de boa resolução que possa gerar um sinal de áudio bom o suficiente para ser avaliado por uma rede treinada ou qualquer outra técnica.

O emprego de processamento de sinais de vídeo também é uma opção interessante para a detecção de choro e para o monitoramento da respiração da criança. Com a câmera posicionada de uma forma que capture as expressões faciais do bebê, é possível identificar as expressões apresentadas pelo bebê (como no trabalho de Lobo et al. [20]) e também auxiliar na detecção de choro, quando o sinal de vídeo é utilizado em adição ao sinal de áudio. As expressões do rosto do bebê durante o choro são bastante características. Então é possível que mesmo sem um processamento complexo de áudio, o sistema seja capaz de detectar um evento de choro baseado apenas na expressão facial e nos altos níveis sonoros detectados pelo sensor de ruído ambiental. Outro uso interessante de câmeras de vídeo é para a captura de imagens do bebê e a "subtração de pixels" entre quadros a fim de detectar movimento na imagem. Enquanto o bebê respira normalmente, a câmera detecta alterações nos pixels entre quadros de vídeo que indicam a movimentação de expansão e contração da caixa torácica da criança. Se há uma parada respiratória (como em SMSL), o sistema para de detectar os movimentos e, então, alerta os pais. Este método é utilizado por Hussain et al. [15].

O monitoramento baseado em processamento de vídeo, apesar de bastante interessante, representa um grande aumento na utilização de recursos de rede para a transferência destes dados para a nuvem. Considerando que a resolução deve ser relativamente alta para que as detecções funcionem, a banda utilizada para a transmissão dos dados em tempo real teria que ser bastante

grande. Na maioria dos trabalhos relacionados, como os trabalhos de Lobo et al. [20] e Hussain et al.[15], não há emprego de computação em nuvem, o que simplifica o funcionamento do sistema em termos de redes e uso de banda.

Outra melhoria interessante seria a detecção de colchão molhado, realizada por Goyal e Kumar [18]. A troca de lençol ou colchão iria auxiliar os pais na prevenção da proliferação de fungos e bactérias. Para isso, utilizaria-se um simples sensor de umidade normalmente utilizado em solos.

Outra característica interessante que poderia ser adicionada é a automação de ferramentas que auxiliam a acalmar o bebê, como por exemplo, o balanço do carrinho, canções de ninar, acionamento de ar-condicionado, etc. Alguns desses acionamentos podem causar o efeito inverso, pois não sabe-se qual o motivo da agitação enquanto o bebê não é avaliado pessoalmente e um carrinho se mexendo sem parar pode causar náuseas no bebê e uma música pode irritá-lo ainda mais. Portanto, essas ferramentas automatizadas devem ser adicionadas com muito cuidado.

Estas são algumas melhorias que podem vir a ser incorporadas ao sistema no futuro. Graças às aplicações distribuídas em contêineres, funcionalidades podem ser adicionadas e removidas livremente. Apesar dos testes terem apresentados bons resultados, acreditamos que com métodos de detecção mais complexos e precisos, pode-se alcançar um desempenho ainda melhor.

Referências Bibliográficas

[1] BOSCH. *Cross-Domain Development Kit XDK110 Platform for Application Development*. [S.l.], 8 2017. 2.0.

[2] BOSCH. *Memory Management*. Disponível em: <https://developer.bosch.com/web/xdk/memory-management>. Acesso em: 9 de mai. de 2021.

[3] BOSCH. *Datasheet: BME280 Digital humidity, pressure and temperature sensor*. [S.l.], 2020.

[4] ESPRESSIF. *Datasheet: ESP32 Series*. [S.l.], 3 2021. V3.6.

[5] MAXIM. *Datasheet: Microphone Amplifier with AGC and Low-Noise Microphone Bias*. [S.l.], 6 2009. Rev 2.

[6] MOUSER ELETRONICS. *Datasheet: HC-SR501 PIR MOTION DETECTOR*. [S.l.].

[7] WEAVEWORKS. *A Practical Guide to Choosing between Docker Containers and VMs*. 2020. Disponível em: <https://www.weave.works/blog/a-practical-guide-to-choosing-between-docker-containers-and-vms>. Acesso em: 1 de mai. de 2021.

[8] CONTROL, C. for D.; PREVENTION. *About SUID and SIDS*. Disponível em: <https://www.cdc.gov/>. Acesso em: 18 de mai. de 2021.

[9] MITCHELL, E. A. International trends in postneonatal mortality. *Archives of Disease in Childhood*, BMJ Publishing Group Ltd, v. 65, n. 6, p. 607–609, 1990. ISSN 0003-9888.

[10] CARLIN, R. F.; MOON, R. Y. Risk Factors, Protective Factors, and Current Recommendations to Reduce Sudden Infant Death Syndrome: A Review. *JAMA Pediatrics*, v. 171, n. 2, p. 175–180, 02 2017. ISSN 2168-6203.

[11] NUNES, M. L. et al. Síndrome da morte súbita do lactente: aspectos clínicos de uma doença subdiagnosticada. *Jornal de Pediatria*, SciELO, v. 77, p. 29 – 34, 02 2001. ISSN 0021-7557.

[12] MESSAOUD, A.; TADJ, C. A cry-based babies identification system. In: *Image and Signal Processing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 192–199. ISBN 978-3-642-13681-8.

[13] LIU, X. et al. Video-based iot baby monitor for sids prevention. In: *2017 IEEE Global Humanitarian Technology Conference (GHTC)*. San Jose, CA, USA: IEEE, 2017. p. 1–7.

[14] DUBEY, Y. K.; DAMKE, S. Baby monitoring systemusing image processingand iot. *International Journal of Engineering and Advanced Technology (IJEAT)*, v. 8, p. 4961–64, 08 2019. ISSN 2249 –8958.

[15] HUSSAIN, T. et al. Intelligent baby behavior monitoring using embedded vision in IoT for smart healthcare centers. *Journal of Artificial Intelligence and Systems*, v. 1, n. 1, p. 110–124, 2019.

[16] ZAKARIA, N. A.; SALEH, F. N. B. M.; RAZAK, M. A. A. Iot (internet of things) based infant body temperature monitoring. In: IEEE. *2018 2nd international conference on biosignal analysis, processing and systems (ICBAPS)*. Kuching, Malaysia: IEEE, 2018. p. 148–153.

[17] MANDKE, S. et al. Iot based infant health monitoring system. *International Journal of Engineering and Technology (IRJET)*, v. 5, p. 3418–3421, 2018.

[18] GOYAL, M.; KUMAR, D. Automatic e-baby cradle swing based on baby cry. *International Journal of Computer Applications*, Citeseer, v. 71, n. 21, p. 39–43, 2013.

[19] ISHAK, D. N. F. M.; JAMIL, M. M. A.; AMBAR, R. Arduino based infant monitoring system. *IOP Conference Series: Materials Science and Engineering*, IOP Publishing, v. 226, p. 012095, aug 2017.

[20] LOBO, C. et al. Infant care assistant using machine learning, audio processing, image processing and iot sensor network. In: IEEE. *2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)*. Coimbatore, India: IEEE, 2020. p. 317–322.

[21] ESPRESSIF. *ESP32. A feature-rich MCU with integrated Wi-Fi and Bluetooth connectivity for a wide-range of applications*. Disponível em: <https://www.espressif.com/en/products/socs/esp32>. Acesso em: 3 de abr. de 2021.

[22] RABBITMQ. *RabbitMQ is the most widely deployed open source message broker*. Disponível em: <https://www.rabbitmq.com/>. Acesso em: 25 de abr. de 2021.

[23] W3C. *Web Services Architecture*. 2004. Relationship to the World Wide Web and REST Architectures. Disponível em: <https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/relwwwrest>. Acesso em: 5 de mai. de 2021.

[24] PALLETS. *Flask. Web development, one drop at a time*. Disponível em: <https://flask.palletsprojects.com/en/2.0.x/>. Acesso em: 3 de abr. de 2021.

[25] TELEGRAM. *Telegram. Uma nova era de mensagens*. Disponível em: <https://telegram.org/>. Acesso em: 2 de mai. de 2021.

[26] ASHTON, K. *That 'Internet of Things' Thing*. 2009. Disponível em: <https://www.rfidjournal.com/that-internet-of-things-thing>. Acesso em: 2 de dez. de 2020.

[27] SURESH, P. et al. A state of the art review on the internet of things (IoT) history, technology and fields of deployment. In: *2014 International Conference on Science Engineering and Management Research (ICSEMR)*. Chennai, Índia: IEEE, 2014. p. 1–8.

[28] CHEN, H.; JIA, X.; LI, H. A brief introduction to iot gateway. In: *IET international conference on communication technology and application (ICCTA 2011)*. Beijing: IET, 2011. p. 610–613.

[29] AMAZON. *O que é a computação em nuvem?* Disponível em: <https://aws.amazon.com/pt/what-is-cloud-computing/>. Acesso em: 1 de mai. de 2021.

[30] AMAZON. *Amazon Web Services (AWS) — Serviços de computação em nuvem.* Disponível em: <https://aws.amazon.com/pt/>. Acesso em: 18 de mai. de 2021.

[31] MICROSOFT. *Serviços de computação em nuvem — Microsoft Azure*. Disponível em: <https://azure.microsoft.com/pt-br/>. Acesso em: 18 de mai. de 2021.

[32] GOOGLE. *Serviços de computação em nuvem | Google Cloud*. Disponível em: <https://cloud.google.com/>. Acesso em: 17 de mai. de 2021.

[33] FREERTOS. *What is An RTOS?* Disponível em: <https://www.freertos.org/about-RTOS.html>. Acesso em: 22 de abr. de 2021.

[34] KUROSE, J.; ROSS, K. Camada de aplicação. In: ____. *Redes de computadores e a Internet: uma abordagem top-down*. 6. ed. [S.l.]: Pearson Education do Brasil, 2014. cap. 2, p. 72–85.

[35] FIELDING, R. T. *Architectural styles and the design of network-based software architectures*. Tese (Doutorado) — Inf. Comput. Sci., Univ. California, Irvine, CA, USA, 2000.

[36] CROCKFORD, D. *ECMA-404 The JSON Data Interchange Standard*. 2017.

[37] DUA, R.; RAJA, A. R.; KAKADIA, D. Virtualization vs containerization to support paas. In: *2014 IEEE International Conference on Cloud Engineering*. Boston, MA, USA: IEEE, 2014. p. 610–614.

[38] DOCKER ENGINE. *Docker Engine overview*. Disponível em: <https://docs.docker.com/engine/>. Acesso em: 30 de abr. de 2021.

[39] IBM. *Containerization*. 2019. Disponível em: <https://www.ibm.com/cloud/learn/containerization>. Acesso em: 21 de abr. de 2021.

[40] RABBITMQ. *AMQP 0-9-1 Model Explained*. Disponível em: <https://www.rabbitmq.com/tutorials/amqp-concepts.html>. Acesso em: 29 de abr. de 2021.

[41] BOSCH. *Enable all Sensors in one*. Disponível em: <https://developer.bosch.com/web/xdk/enable-all-sensors-in-one>. Acesso em: 5 de mai. de 2021.

[42] BOSCH. *Developer Portal*. Disponível em: <https://developer.bosch.com>. Acesso em: 2 de dez. de 2020.

[43] BOSCH. *Datasheet: Cross Domain Development Kit | XDK*. [S.l.], 11 2017.

[44] MAXIM INTEGRATED. *Datasheet: MAX44009 Industry's Lowest-PowerAmbient Light Sensor with ADC*. [S.l.], 2011.

[45] PALLETS. *Flask. Web development, one drop at a time*. Disponível em: <https://flask-sqlalchemy.palletsprojects.com/en/2.x/>. Acesso em: 3 de abr. de 2021.

[46] ORACLE. *MySQL*. Disponível em: <https://www.mysql.com/>. Acesso em: 19 de mai. de 2021.

[47] GRAFANA LABS. *Grafana: The open observability platform | Grafana Labs*. Disponível em: <https://grafana.com/>. Acesso em: 19 de mai. de 2021.

[48] DOCKER. *Empowering App Development for Developers | Docker*. Disponível em: <https://www.docker.com/>. Acesso em: 17 de mai. de 2021.

[49] HUNT-WALKER, N. *An introduction to the Flask Python web app framework*. 2018. Disponível em: <https://opensource.com/article/18/4/flask>. Acesso em: 25 de abr. de 2021.

[50] PYTHONBASICS. *What is Flask Python*. Disponível em: <https://pythonbasics.org/what-is-flask-python/>. Acesso em: 25 de abr. de 2021.

[51] SQLALCHEMY. *The Python SQL Toolkit and Object Relational Mapper*. Disponível em: <https://www.sqlalchemy.org/>. Acesso em: 26 de abr. de 2021.

[52] MILANI, A. *MySQL: Guia do Programador*. [S.l.]: Novatec, 2006.

[53] DOCKER. *Docker Hub*. Disponível em: <https://hub.docker.com/>. Acesso em: 17 de mai. de 2021.

[54] WIRESHARK. *Wireshark - Go Deep*. Disponível em: <https://www.wireshark.org/>. Acesso em: 18 de mai. de 2021.

[55] ARDUINO. *Arduino IDE*. Disponível em: <https://www.arduino.cc/en/software>. Acesso em: 3 de abr. de 2021.

[56] ARDUINO. *WiFi library*. Disponível em: <https://www.arduino.cc/en/Reference/WiFi>. Acesso em: 3 de abr. de 2021.

[57] BLANCHON, B. *ArduinoJson. The best JSON library for embedded C++*. Disponível em: <https://arduinojson.org/>. Acesso em: 15 de abr. de 2021.

[58] DOCKER. *Overview of Docker Compose*. Disponível em: <https://docs.docker.com/compose/>. Acesso em: 17 de mai. de 2021.

[59] HIGUERA, V. *What Is the Best Room Temperature for Baby?* 2020. Disponível em: <https://www.healthline.com/health/baby/room-temperature-for-babymaintaining-temperature>. Acesso em: 4 de mai. de 2021.

[60] THELULLABYTRUST. *The safest room temperature for babies*. Disponível em: <https://www.lullabytrust.org.uk/safer-sleep-advice/baby-room-temperature/>. Acesso em: 4 de mai. de 2021.

[61] ENVIRONMENTAL PROTECTION AGENCY. *Indoor Air Quality Tools for Schools: Reference Guide*. [S.l.], 1 2009.

[62] TORRES, R.; BATTAGLINO, D.; LEPAULOUX, L. Baby cry sound detection: A comparison of hand crafted features and deep learning approach. In: SPRINGER. *International Conference on Engineering Applications of Neural Networks*. Cham: Springer International Publishing, 2017. p. 168–179.

Apêndice A

Operação do Gateway

```
version: '3.7'
networks:
    net2:
        name: net2

services:

    prod:
        container_name: prod
        ports:
            - 54321:5000
        build:
            context: ./prod
        restart: unless-stopped
        networks:
            - net2

    cons:
        container_name: cons
        build:
            context: ./cons
        restart: unless-stopped
        networks:
            - net2

    rabbitmq:
        image: rabbitmq
        networks:
            - net2

```

Listagem A.1: Arquivo docker-compose.yml do Gateway

```
    @app.route('/esp32', methods=['POST'])
    def post_sound():
        connection = pika.BlockingConnection(pika.ConnectionParameters('
    rabbitmq'))
        channel = connection.channel()

        channel.exchange_declare(exchange="main_ex", exchange_type="direct")

        channel.basic_publish(exchange='main_ex',
                              routing_key='esp32',
                              body=request.data)

        channel.close()
        return request.data, status.HTTP_201_CREATED


    @app.route('/xdk', methods=['POST'])
    def post_xdk():
        connection = pika.BlockingConnection(pika.ConnectionParameters('
    rabbitmq'))
        channel = connection.channel()

        channel.exchange_declare(exchange="main_ex", exchange_type="direct")

        channel.basic_publish(exchange='main_ex',
                              routing_key='xdk',
                              body=request.data)

        channel.close()
        return request.data, status.HTTP_201_CREATED

```

Listagem A.2: Endpoints da API Produtora - Gateway

```
    def main():
        connection = pika.BlockingConnection(pika.ConnectionParameters('
    rabbitmq'))
        channel = connection.channel()

        channel.exchange_declare(exchange="main_ex", exchange_type="direct")

        result = channel.queue_declare(queue='esp32', exclusive=True)

        queue_name = result.method.queue

        def callback_esp32(ch, method, properties, body):
            print(" [x] Received %r" % body)
            requests.post("http://{0}:{1}/esp32".format(ip_addr, port_num),
```

```
    data=body.decode(),
                          headers={"Content-Type": "application/json"})

        channel.queue_bind(exchange='main_ex', queue=queue_name,
                           routing_key='esp32')

        channel.basic_consume(
            queue=queue_name, on_message_callback=callback_esp32, auto_ack=
    True)

        result2 = channel.queue_declare(queue='xdk', exclusive=True)

        queue_name2 = result2.method.queue

        def callback_xdk(ch, method, properties, body):
            print(" [x] Received %r" % body)
            requests.post("http://{0}:{1}/xdk".format(ip_addr, port_num),
    data=body.decode(),
                          headers={"Content-Type": "application/json"})

        channel.queue_bind(exchange='main_ex',
                           queue=queue_name2, routing_key='xdk')

        channel.basic_consume(
            queue=queue_name2, on_message_callback=callback_xdk, auto_ack=
    True)

        channel.start_consuming()
        print("Program Started. Consuming...")

```

Listagem A.3: Função principal da aplicação Consumidora

Apêndice B

Operação do Backend

```
version: '3.7'
networks:
    net1:
        name: net1

services:
    mysql:
        container_name: mysql
        build:
            context: ./db
        restart: unless-stopped
        networks:
            - net1
        volumes:
            - ./dbdata:/var/lib/mysql
        command: mysqld --default-authentication-plugin=
mysql_native_password --skip-mysqlx
        cap_add:
            - SYS_NICE # CAP_SYS_NICE

    app:
        container_name: app
        ports:
            - 54322:5000
        build:
            context: ./app
        restart: unless-stopped
        networks:
            - net1
        depends_on:
            - mysql

```

```
    grafana:
        container_name: grafana
        user: "$UID:$GID"
        ports:
            - 3000:3000
        networks:
            - net1
        build:
            context: ./grafana
        volumes:
            - ./grafana-data:/var/lib/grafana


volumes:
    dbdata:

```

Listagem B.1: Arquivo docker-compose.yml do módulo de Recebimento e Armazenamento de dados

```

import telebot
import urllib
import json
import requests

bot = telebot.TeleBot("1727576179:AAE6t-psMqxv3uNd9_yP62JbG-qLWCykKpg")
chat_id = -555998562
ip_addr = "xx.xx.xx.xx"
port_num = 54322


@bot.message_handler(commands=['start', 'help'])
def send_start_message(message):
    bot.reply_to(message, "Ola, eu sou o Bot monitor de bebe.")


@bot.message_handler(commands=['environment', 'env'])
def send_environment_message(message):
    rt = requests.get(
        url="http://{0}:{1}/temperature/now".format(ip_addr, port_num))
    rh = requests.get(
        url="http://{0}:{1}/humidity/now".format(ip_addr, port_num))
    rl = requests.get(url="http://{0}:{1}/light/now".format(ip_addr,
port_num))

    rt_data = rt.json()
```

```
27          rh_data = rh.json()
28          rl_data = rl.json()
29
30          msg = "Temperatura atual: {0}  C \nUmidade atual: {1}%\nLuminosidade
    atual: {2} mlx".format(
31              round(rt_data[0]['temperature']/1000, 1), rh_data[0]['humidity'],
     rl_data[0]['light'])
32
33          bot.reply_to(message, msg)
34
35
36      bot.polling()
37
```

Listagem B.2: Código da aplicação do contêiner *bot*