UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

EDUARDO LINO XIMENES
MATHEUS BIAGGIO CHWESZCZUK

DESENVOLVIMENTO DE UMA ARQUITETURA IOT PARA SENSORIAMENTO DE AVIÁRIOS

CAMPO MOURÃO
2022

EDUARDO LINO XIMENES
MATHEUS BIAGGIO CHWESZCZUK

DESENVOLVIMENTO DE UMA ARQUITETURA IOT PARA SENSORIAMENTO DE AVIÁRIOS

Development of an iot architecture for avian sensing

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná (UTFPR).
Orientador(a): Professor Dr. Marcio Rodrigues da Cunha.

CAMPO MOURÃO

2022

EDUARDO LINO XIMENES
MATHEUS BIAGGIO CHWESZCZUK

DESENVOLVIMENTO DE UMA ARQUITETURA IOT PARA SENSORIAMENTO DE AVIÁRIOS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 09/junho/2022

_____

Gilson Junior Schiavon
Doutorado em Engenharia Química
Universidade Tecnológica Federal do Paraná

_____

Roberto Ribeiro Neli
Doutorado em Engenharia Elétrica
Universidade Tecnológica Federal do Paraná

_____

Marcio Rodrigues da Cunha
Doutorado em Engenharia Elétrica
Universidade Tecnológica Federal do Paraná

CAMPO MOURÃO

2022

RESUMO

Uma das tendências crescentes no avanço da indústria 4.0 é a possibilidade da consulta de dados em tempo real dos sistemas produtivos. Processos de extrema importância podem ser supervisionados remotamente para ter certeza de que os parâmetros estão em níveis ótimos, além de ter a capacidade de avaliar possíveis falhas na operação. Este projeto tem como objetivo realizar o monitoramento dos principais parâmetros para o funcionamento adequado de uma granja: a temperatura, umidade e luminosidade, os quais tem grande impacto no crescimento e no desenvolvimento dos animais. Para a leitura destes sensores foi utilizado um microcontrolador ESP32, o qual realiza a coleta dos dados e a comunicação através da rede WIFI com o banco de dados. Para o armazenamento destes dados foi utilizado um banco de dados Firebase. Os dados foram exibidos ao usuário através de um aplicativo mobile desenvolvido em *React-Native*, possibilitando que os parâmetros sejam consultados tanto em tempo real quanto ao seu histórico das últimas 24 horas. Com base nestes conceitos, o sistema foi projetado, desenvolvido e testado em ambiente controlado, resultando em uma solução completa para a coleta, processamento, armazenamento e exibição ao usuário. O sistema funcionou de acordo, demonstrando ser uma solução viável para o monitoramento em tempo real de uma granja industrial, trazendo melhor segurança, eficiência e maior produtividade para o produtor de frango.

Palavras-chave: sensores; internet das coisas; Firebase; React-Native.

ABSTRACT

One of the growing trends in the advancement of industry 4.0 is the possibility of querying data in real-time from production systems. Processes can be remotely supervised to make sure parameters are at optimal levels, and have the ability to assess possible failures in operation. This project aims to monitor the main parameters for the proper functioning of a farm: temperature, humidity and luminosity, which has a great impact on the growth and development of animals. An ESP32, microcontroller was used to read these sensors, which collects data and communicates through the WIFI network with the database. A Firebase database will be used to store this data. The data will be displayed to the user through a mobile application developed in React Native, allowing the parameters to be consulted in real-time or in the history of the last 24 hours. Based on these concepts, the system was designed, developed and tested in a controlled environment, resulting in a complete solution for collection, processing, storage and display to the user. The system worked as expected, proving to be a viable solution for real-time monitoring of an industrial farm, bringing better safety, efficiency and greater productivity to the chicken producer.

Keywords: sensors; internet of things; Firebase; React-Native.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	Rotina e padrões de programação
IDE	Ambiente de desenvolvimento integrado
IOT	Internet das coisas
IP3	Protocolo de comunicação
JSON	Extensão de arquivos Javascript
LDR	Resistencia sensível a luz
NTC	Termistor de coeficiente negativo de temperatura
PTC	Termistor de coeficiente positivo de temperatura
URL	Localizador uniforme de recursos
USB	Porta universal
WSN	Rede de comunicação de sensores

LISTA DE SÍMBOLOS

Vcc	Tensão em corrente contínua
GND	Terra

SUMÁRIO

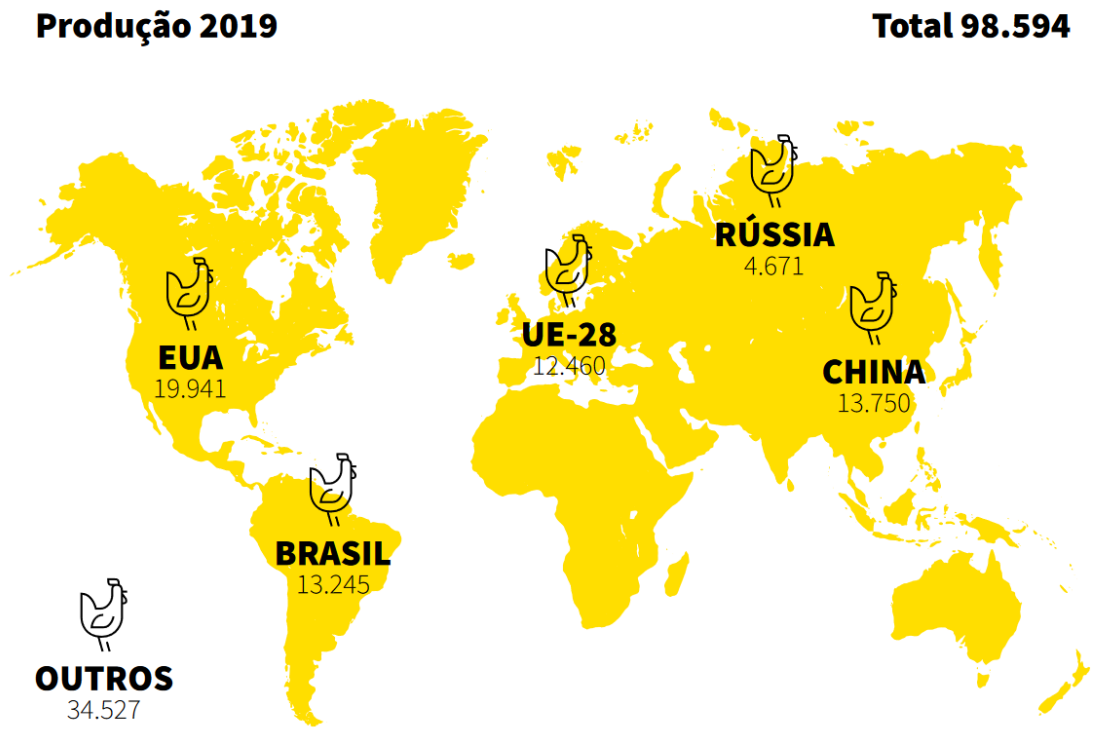
1 INTRODUÇÃO

A produção de alimentos para um país são fatores que impactam diretamente na sociedade, gerando renda e empregos diretos e indiretos. As carnes de frango e ovos são fundamentais para garantir a segurança alimentar nas cidades, sendo proteínas muito consumidas no Brasil (ABPA, 2020).

Com o grande crescimento da produção avícola, devido ao aumento do consumo de carne e ovos em uma escala mundial, cria-se uma necessidade de melhorias constantes em produtos e sistemas que tem impacto direto na produção e no bem-estar do animal (BELLAVER, 2003).

Segundo o relatório anual da ABPA (Associação Brasileira de Proteína Animal), em 2019, o Brasil foi o terceiro maior produtor mundial de carne de aves com 13,2 milhões de toneladas, perdendo apenas para CHINA e EUA. Também apontou que, em 2019, foi o país que mais exportou carne de frango mundialmente com, aproximadamente, 4,2 milhões de toneladas de carnes exportadas.

Figura 1 - Mapa maiores produtores de carne aviária mundial



Fonte: ABPA (2020, p.32)

Tinôco (2001) diz que a Avicultura é o setor agropecuário que tem uma das tecnologias mais avançadas e em maior quantidade, devido ao grande progresso em genética, nutrição, manejo e sanidade. Com o aumento da produção de aves, a exigência de conforto térmico ambiental se tornou ainda maior, portanto, novas tecnologias se tornam necessárias.

As aves têm uma temperatura corporal por volta de 41 ºC e 42 ºC, comparando com um mamífero, esse valor é bem superior. Em sua estrutura biológica, as aves, geralmente, são desprovidas de glândulas sudoríparas, ou seja, não são capazes de realizar uma transpiração que consiga amenizar a temperatura corporal. Sabendo disso, o controle térmico de uma avícola afeta diretamente no resultado final do desempenho dos animais (WELKER; ROSA; MOURA; MACHADO; CATELAN; UTTPATEL, 2008).

De acordo com a pesquisa de Oliveira (2006), com população de 149 pintos machos, com a temperatura e a humidade relativa do ar altas podem afetar diretamente os cortes nobres.

Welker, Rosa, Moura, Machado, Catelan e Uttpatel (2008) ainda afirmam que pesquisas validam o fato de a temperatura do ambiente agir diretamente na temperatura corporal da ave.

Graças a modernização na agricultura, ficou cada vez mais comum substituir o trabalho manual, que muitas vezes pode-se conter uma grande quantidade de falhas, em rotinas automatizadas de equipamentos microcontrolados. Isso proporciona uma diminuição no valor de tais produtos (SANTOS *et al.*, 2014).

Com a chegada da Internet das Coisas, os ambientes se tornaram cada vez mais automatizados criando uma conectividade muito grande entre os seres humanos e máquinas. Essa comunicação permite trocas de informações e tomadas de decisões automáticas para aumentar a produtividade e reduzir custos (SANTOS *et al.*, 2014).

O crescente aumento do mercado aviário no Brasil, fez com que os produtores invistam cada vez mais. Esse investimento torna o produto final cada vez melhor em relação a qualidade. Porém, a rentabilidade depende muito da forma em que se cria e maneja as aves (SANTOS *et al.*, 2014).

Desta forma, este trabalho pretende contribuir com esta importante atividade econômica articulando a aplicação de tecnologias emergentes neste setor produtivo.

1.1 Objetivos

Desenvolver uma aplicação IoT de monitoramento de aviários convencionais em tempo real com dados armazenados em um banco de dados virtual.

Os objetivos específicos do trabalho são:

- Desenvolver uma aplicação com sistema *Android* para mostrar os valores monitorados em tempo real;
- Desenvolver um sistema de monitoramento com sensores de Temperatura, Umidade e Luminosidade;
- Criar um banco de dados para armazenamento de dados monitorados;
- Testar e avaliar o desempenho de todo o sistema em conjunto.

1.2 Justificativa

Muitos problemas podem ocorrer na criação de frango de corte que são influenciados por fatores como temperatura, umidade, luminosidade, entre outros, e podem ser suscetíveis a estresse térmico e fraturas em sua fase inicial de vida e desenvolver problemas que não são normalizados até o abate, como deficiência de crescimento e fraturas (AMARAL *et al.*, 2011).

Visto isso, a estabilização da temperatura e outros fatores em ambientes de produção são necessários para que o frango consiga alcançar o seu máximo desempenho no crescimento.

Existem muitas exigências do mercado consumidor de produtos de origem animal referentes a questão ambiental, segurança alimentar e bem-estar animal. Essas questões passam pelo conforto térmico dos ambientes de criação das aves (aviários), sendo um dos fatores que afetam diretamente o desempenho das aves. Com isso, o monitoramento constante das condições climáticas e outros fatores que influenciam o crescimento animal se torna uma obrigação do produtor nos dias atuais.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão abordados os levantamentos teóricos que foram utilizados para auxiliar na elaboração da arquitetura e no estudo do caso, as tecnologias utilizadas e as vantagens de utiliza-las.

2.1 Internet das coisas

A Internet das Coisas (do inglês *Internet of Things*) se manifestou devido aos grandes avanços tecnológicos de várias áreas como Sistemas Embarcados, Microeletrônica, Comunicação e Sensoriamento. A IoT pode ser definida como uma extensão da Internet para objetos do dia-a-dia, o que proporciona que esses objetos se conectem na internet. Esta conexão tem inúmeros benefícios como, por exemplo, possibilitar que o objeto seja controlado remotamente e acessados com provedores de serviços, criando inúmeras oportunidades industriais e acadêmicas (SANTOS, 2016).

Com esse avanço da tecnologia, é notável ver que a população está cada vez mais priorizando aparelhos *mobiles* à computadores *desktop*. É possível ver tais dados através das pesquisas realizadas no Brasil (IBGE, 2014) e nos Estados Unidos da América (LELLA e LIPSMAN, 2014). Com isso, é possível deduzir, que tais aparelhos estão virando cada vez mais uma necessidade diária (CARRION; QUARESMA, 2019).

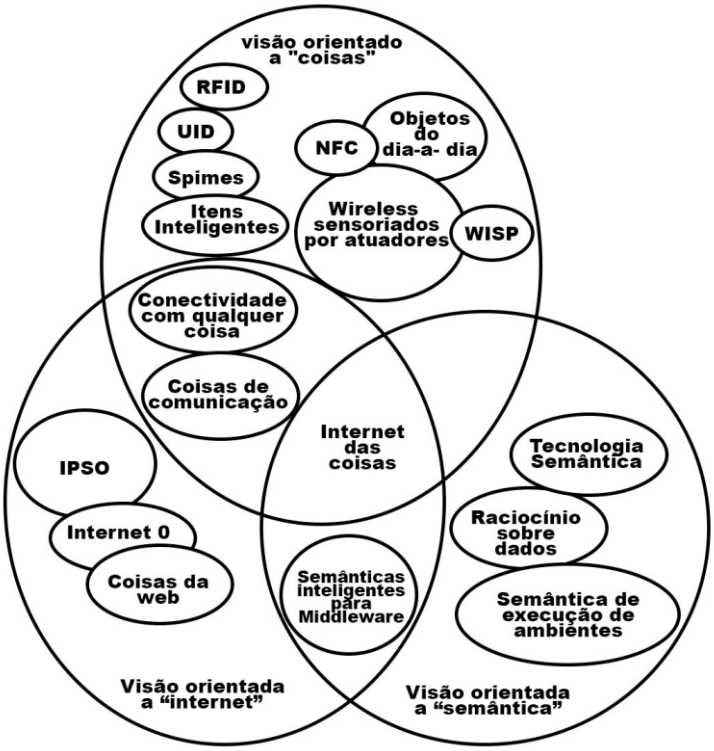
Swamy, Jadhav e Kulkarni (2017) entra em acordo com Moon (2016) ao mencionar que a estrutura do funcionamento de uma IoT consiste em uma comunicação de rede e protocolos, que são responsáveis por criar um caminho e uma transmissão de dados entre os diferentes aparelhos conectados. Em síntese, a IoT é um conjunto de objetos físicos interconectados através de um endereço IP3 ou outra rede, formando um ecossistema. Essa conexão os deixa capaz de trocar, armazenar e coletar dados (MOON, 2016).

A IoT fornece um grande volume de dados que impacta em determinadas aplicações por possuírem informações muito úteis. Para garantir a confiabilidade e a segurança de tais dados, a estrutura da IoT é baseada em três camadas, sendo elas a camada de percepção (sensoriamento), a camada de rede (transferência de dados) e a camada de aplicação (armazenamento e manipulação de dados). Para a execução de tal estrutura, é necessário utilizar algoritmos capazes de fazer a interconexão, deixando-o bastante complexo para a execução em uma rede de sensores sem fio (WSN) de baixa energia. Todavia, na conexão IoT, todos os objetos estão interconectados através da rede, com isso é possível deixar a sobrecarga

computacional para ser processado na nuvem ou ser distribuída entre vários dispositivos (OLIVEIRA, 2019).

Como identificado por Atzori *et al.* (2010), a Internet das Coisas pode ser realizada em três paradigmas – *internet-oriented*, *things-oriented* (sensores) e *semantic-oriented* (conhecimento). Essa definição pode ser vista na Figura 2.

Figura 2 - Internet das coisas.



Fonte: Adaptado de ATZORI *et al.* (2010, p.3)

A partir dos três paradigmas e da Figura 2, Atzori *et al.* (2010) considera:

- Orientada a objetos (coisas): O objeto (sensor) e como identificá-lo e implementá-lo;
- Orientada à internet: A evolução nessa nova visão, de inclusão de uma infraestrutura global que conecta objetos genéricos físicos;
- Orientada a semântica (conhecimento): Se baseia em como representar, armazenar, interconectar e organizar as informações geradas pela IoT. É a exploração de soluções e modelagens adequadas, descrevendo com raciocínio os dados gerados pela IoT.

De acordo com Gubbi et al. (2013), todas essas definições de IoT são necessárias devida à natureza interdisciplinar do assunto. A IoT pode ser desencadeada em algum domínio de aplicação onde os três paradigmas se cruzam.

2.2 Sensores e atuadores

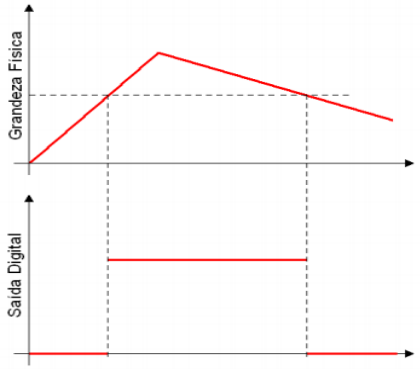
Thomazini e Albuquerque (2009) tem o mesmo pensamento que Brito (2017), ao afirmarem que atuadores “são dispositivos que modificam uma variável controlada. Recebem um sinal proveniente do controlador e agem o sistema controlado.” (THOMAZINI E ALBUQUERQUE, 2009, p. 17), ou seja, são instrumentos que fazem o papel intermediário de um sistema, capaz de transformar uma grandeza em outra. Brito (2017) cita o exemplo de que são utilizado transdutores em subestações para converter sinais que estão nas grandezas de centenas de volts em sinais de tensão ou corrente.

Já os sensores são dispositivos sensíveis a algum tipo de energia do ambiente. Essas energias podem ser luminosas, térmica e cinética. Todas essas energias têm algo em comum, são grandezas que geram informações capazes de serem medidas, tais como: temperatura, pressão, velocidade, corrente, aceleração etc (THOMAZINI; ALBUQUERQUE, 2009). Além disso, sensores estão em contato direto com a variável controlada, que ao captar o sinal mensurado, envia sinais para outros subsistemas (BRITO, 2017).

Quando usados em sistemas que necessitam de controle, nem sempre são encontradas as características elétricas necessárias para a utilização. De acordo com Wendling (2010), normalmente o sinal de saída deve ser manipulado antes da sua leitura no sistema de controle. Isso geralmente é realizado com um circuito de interface para a produção de um sinal que possa ser lido pelo controlador.

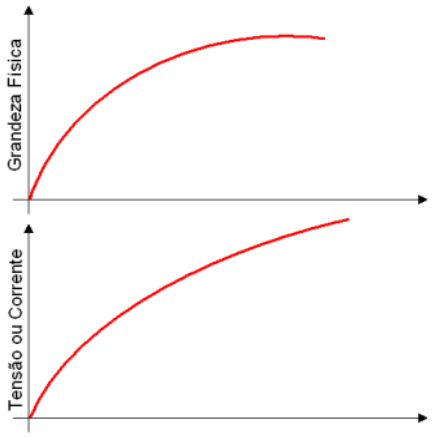
As saídas podem ser digitais ou analógicas. Na digital, a saída é discreta e só assumem valores “0” ou “1” (Figura 3) e na analógica, a saída é contínua (Figura 4).

Figura 3 - Saída digital



Fonte: WENDLING (2010, p.6)

Figura 4 - Saída analógica



Fonte: WENDLING (2010, p.6)

Neste contexto, o sistema irá monitorar as seguintes grandezas físicas, luminosidade, temperatura e umidade. Os conceitos sobre a medição de tais grandezas físicas são apresentadas a seguir.

2.2.1 Fotorresistor

O sensor mais utilizado em projetos que se faz necessário o sensoriamento de alteração de luminosidade, é o fotorresistor. Ele é constituído pôr sulfeto de cádmio, o qual é chamado de fotocélulas de sulfeto de cádmio. Além disso, contém dois terminais, fazendo que seja possível conectar as duas extremidades das fotocélulas de sulfeto de cádmio no sistema (THOMAZINI e ALBUQUERQUE, 2010).

Tais fotocélulas de sulfeto de cádmio, tem uma resistência muito grande quando não ocorre nenhuma incidência de luz, da ordem de milhões de ohms. Já quando ocorre uma incidência de luz, ocorre uma alteração em sua resistência, ficando na casa de centenas de milhares de ohms. Esse efeito ocorre, pois com a incidência de luz nas fotocélulas de sulfeto de cádmio, ocorre uma liberação dos portadores, ajudando na condução de corrente elétrica (THOMAZINI e ALBUQUERQUE, 2010).

Sabendo do seu funcionamento, é possível entender que não existe polarização em seus terminais, ou seja, a corrente pode circular nos dois sentidos, não tendo divergência do resultado em ambos os sentidos (THOMAZINI e ALBUQUERQUE, 2010).

2.2.2 Termistor

Para a detecção automática, medição e controle da temperatura em um sistema, termistores são os sensores mais utilizados. Este tipo de sensor é sensível a variação de temperatura, variando a sua resistência elétrica (THOMAZINI e ALBUQUERQUE, 2010).

É possível encontrar dois tipos de termistor, o de coeficiente positivo de temperatura (PTC) e o de coeficiente negativo de temperatura (NTC). O funcionamento dos dois tipos é similar, exceto que no PTC a resistência aumenta de acordo com a temperatura aumenta, já o NTC, o funcionamento é o inverso, a sua resistência aumenta de acordo com a diminuição da temperatura (THOMAZINI e ALBUQUERQUE, 2010).

2.2.3 Sensor de umidade capacitivo

CASTRO (2011) diz em sua pesquisa que não existe uma maneira de realizar a mensuração da umidade do ar de uma forma direta, e sim através de mudanças nas propriedades elétricas, deformação mecânica e outros métodos de alguns materiais.

Nos sensores de umidade, existe um material higroscópico, o qual tem a função de absorver a umidade do ar, sempre deixando em equilíbrio a umidade relativa do meio e a sua (CASTRO, 2011).

O sensor de umidade capacitivo, tem o seu funcionamento semelhante à de um capacitor. Existe dois eletrodos, separados, que contem substrato de retenção de umidade. Quando esses eletrodos absorvem água, ocorre uma alteração da sua capacitância (CASTRO, 2011).

2.3 Microcontrolador

Para a realização de tarefas que necessitariam de um processamento computacional, a utilização de microcontroladores seria praticamente obrigatória. Tais componentes são de pequeno porte e com um custo baixo. O controle remoto de sistemas embarcados se faz uso de tal objeto como o componente que controla todo o sistema. Sua arquitetura é composta, basicamente, por uma unidade processadora, memórias, entradas e saídas, controle temporal e conversores analógicos e digitais (SANTOS; JUNIOR, 2019).

2.3.1 ESP32

O microcontrolador ESP32 foi projetado no ano de 2016 pela empresa *Espressif Systems*. Desde então, graças as suas principais características, como a sua velocidade de processamento, acessibilidade e conectividade, ele é considerado como um dos mais desenvolvidos do mercado (SANTOS; JUNIOR, 2019).

O ESP32 é um *chip* de baixo custo energético, mas que contêm integrado a sua arquitetura um módulo *WIFI* (2,4 GHz) e um outro módulo de *Bluetooth* 4.2. Graças a isso, acrescentando a facilidade de implementação e tendo o custo bem acessível, para um projeto inicial, é uma excelente escolha para se implementar um circuito IoT (KURNIAWAN, 2019).

2.4 Banco de dados

Está cada dia mais difícil de se viver sem ter um contato, que seja o mínimo, com sistemas que utiliza banco de dados. Por exemplo, ao se realizar um depósito em bancos, ao se realizar reservas em hotéis ou restaurantes, muito provável que se esteja utilizando um sistema que utilize banco de dados. Com esses exemplos, é possível ter uma definição genérica e simples do que é um banco de dados, ou seja, banco de dados é uma coleção de dados que os mantem gravados em sua base e o seu significado fica implícito para cada situação (RAMEZ *et al.*, 2005).

Um banco de dados pode ser gerado e mantido manualmente ou automatizado (através de um computador ou um controlador programável). É notório que cada vez mais as empresas e pessoas, em seus usos de interesse próprio, utilizarem um computador para realizar tais atividades (RAMEZ *et al.*, 2005).

Ramez *et al.* (2005) reforça que é cada vez é mais comum as pessoas utilizarem um sistema que gerencia o banco de dados, pois isso facilita e agiliza muito a sua construção.

2.4.1 Firebase

Firebase é um conjunto de estruturas projetado para atender as aplicações empresariais, que se faz necessário do uso da tecnologia em tempo real, focada na tecnologia *mobile* e aplicações *web*. Além disso, quando se utiliza essa tecnologia, não ocorre risco de

perda de dados, isso porque os dados são atualizados de forma singular, ou seja, os dados são atualizados instantaneamente para cada usuário (CHATTERJEE *et al.*, 2018).

O *Firebase* se torna muito atrativo para desenvolvedores, graças a sua facilidade e praticidade. Isso graças a que, o próprio *Firebase* facilita muito o desenvolvimento de aplicativos *web*, pois a própria tecnologia fica encarregada da programação do servidor. Juntamente a isso, oferece uma plataforma básica e unificada com diversas aplicações oferecidas pela *Google* (CHATTERJEE *et al.*, 2018).

Chatterjee *et al.* (2018) explica que nos casos em que se há necessidade de comunicação ou um aplicativo que se há uso de *chat*, é necessário o uso dos seguintes componentes oferecido pelo *Firebase*:

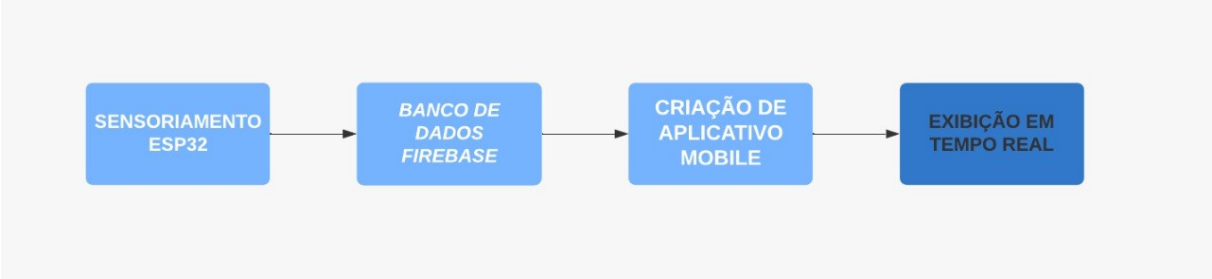
- Base de dados em tempo real: A base de dados em tempo real é hospedada na nuvem. Os dados são armazenados como JSON e sincronizados para cada cliente, atualizando sempre que ocorre uma recepção de um dado novo.
- Autenticação: Autenticação Firebase oferece uma autenticação simples e fácil de se usar. Através do seu serviço *backend*, é possível realizar a autenticação do usuário através de senhas, e-mail, número de telefone etc.
- Armazenamento: O armazenamento *Firebase* é para aplicações que se faz necessidade de se armazenar e fornecer conteúdo gerado pelo usuário, como por exemplo fotos, vídeos etc. Graças a confiabilidade que *Firebase* oferece, não há qualquer risco de se realizar qualquer tipo de *download* ou transferência, independente da qualidade de rede que o usuário está conectado.
- Mensagem da nuvem: É uma multiplataforma que é disponibilizado para os desenvolvedores que possa trocar mensagens de notificação com custo zero e confiáveis.

3 METODOLOGIA

Neste capítulo serão descritos os materiais e métodos utilizados para atingir os objetivos propostos por este trabalho. No diagrama da Figura 5, é apresentado o fluxo simplificado do sistema. Inicialmente, tem-se a configuração da IDE, montagem do circuito e leitura de sensores utilizando o microcontrolador ESP32, para receber esses dados foram utilizados um sensor DHT11 que fornece leituras de Umidade e Temperatura e um sensor LDR que fornece leituras de Luminosidade do ambiente. Com este sensoriamento realizado, os dados serão armazenados em um banco de dados "*Real-Time*" em que sobrescreve os valores lidos em tempo real.

Na próxima etapa foi criado um aplicativo *mobile* que utiliza o sistema *Android* e que se comunica com o banco de dados de duas formas, incialmente fazendo uma autenticação com e-mail cadastrado e leitura dos dados em tempo real no banco e exibe esses valores em tempo real em um painel.

Figura 5 - Diagrama de blocos do sistema completo



Fonte: Autoria própria (2022)

3.1 Sensoriamento ESP32

Com a finalidade de que tenha um resultado satisfatório e otimizado, foi implementado um código que consiga receber os dados dos sensores e que os mesmos sejam enviados para um banco de dados. Para tal, o fluxograma, representado na Figura 6, fica demonstrado, de forma sucinta, todo esse processo.

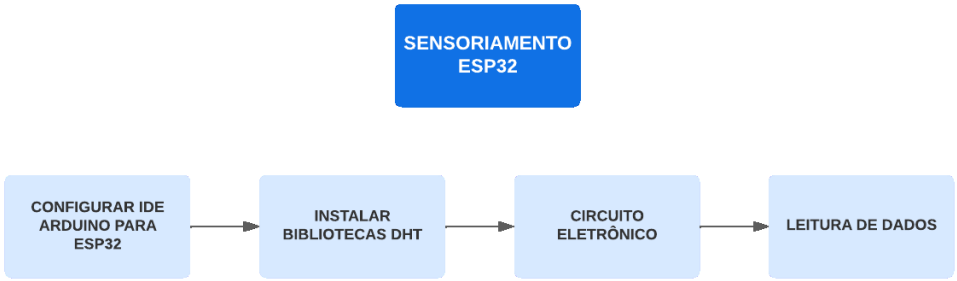
Figura 6 - Fluxograma código ESP32

Fonte: Autoria própria (2022)

No código implementado, houve a necessidade de adicionar bibliotecas já existentes, para que o resultado final seja o mais eficiente e mais claro possível. Após esse passo, foram declaradas variáveis que estariam presentes em todo o decorrer do código. Tendo as bibliotecas certas e as variáveis necessárias declaradas, inicia-se o processo em que o código irá se conectar com uma rede WIFI, para que possa ter uma conexão com a internet e que ocorra todo o envio de dados para o banco de dados. Com a conexão estável, inicia-se o processo do código, que ocorrerá em *loop* infinito, em que será realizado toda a captação dos valores dos sensores e enviado, tais valores, para um banco de dados. É válido notar que todo o processo não tem uma interrupção após a conexão com a internet.

Para ter uma visão mais completa de todo o processo, foi realizado um diagrama de blocos com todo o sistema. Tal diagrama pode ser visto na Figura 7.

Figura 7 - Diagrama de blocos ESP32



Fonte: Autoria própria (2022)

Tendo em vista todo o processo, a seguir será explicado cada processo individualmente em sequência.

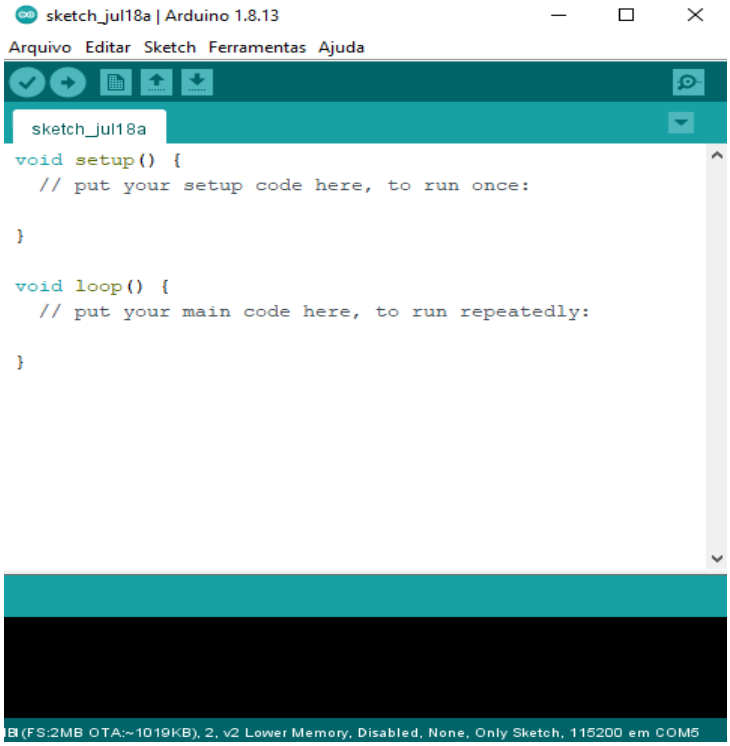
3.1.1 Configurar IDE do Arduino para ESP32

O Arduino, um hardware livre, conta com uma IDE, do *inglês Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado que permite a conexão com várias placas de desenvolvimento. A linguagem de programação é basicamente em C, com algumas funções específicas usadas na plataforma, que auxiliam em rotinas.

Com a IDE instalada e pronta para uso, cria-se então um *sketch*, que é o programa que vai ser utilizado e é possível carregá-lo para um microcontrolador via USB, facilitando o processo de gravação do *chip* e evitando o uso de conversores USB/Serial. Todo esse processo de compilação, *debug*, e carregamento é feito do *software* para o microcontrolador via IDE.

Um *sketch*, antes de ter um código escrito, apresenta em sua estrutura básica, mostrado na Figura 8, as funções "*setup*()" e "*loop*()". A função "*setup*()" é a primeira função a ser executada quando o microcontrolador é ligado, e as instruções nela são executadas apenas uma vez. A função "*setup*()" é onde geralmente são definidas as variáveis e alguns valores base. A função "*loop*()", todas as instruções escritas nesta parte são executadas em repetições infinitas.

Figura 8 - IDE Arduino



Fonte: Autoria própria (2022)

Como visto na Figura 8, a interface é bem simples. O botão ✓ é usado para verificar/compilar o *sketch* e o botão → carrega ao microcontrolador. O sucesso da compilação ou os erros são exibidos na área em preto. A Figura 9 mostra uma tabela que exibe alguns comandos básicos utilizados na programação pela IDE.

Figura 9 - Tabela comandos básicos Arduino

COMANDOS	DESCRIÇÃO
HIGH	Nível lógico alto
LOW	Nível lógico baixo
INPUT	Pino em modo de entrada
OUTPUT	Pino em modo de saída
analogRead()	Faz a leitura de um pino e retorna um valor analógico
digitalRead()	Faz a leitura de um pino e retorna um valor binário
analogWrite()	Saída D/A
digitalWrite()	Saída binária

Seral.begin()	Inicia o monitor da porta seral do microcontrolador
Serial.print()	Exibe caracteres na porta seral
Dealy()	Pausa na execução da rotina em milissegundos

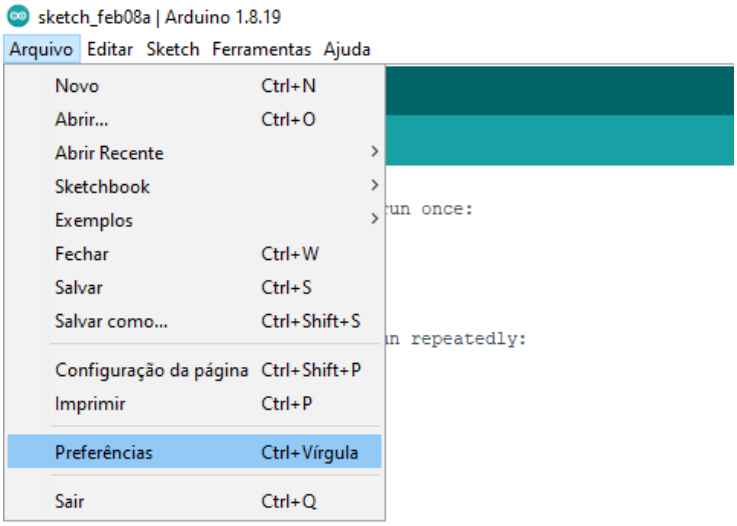
Fonte: Autoria própria (2022)

Para poder utilizar o ESP32 através da IDE do *Arduino*, é necessário acrescentar a biblioteca, já existente no banco de bibliotecas do Arduino. É mandatório realizar uma configuração no IDE do *Arduino* para que o seu funcionamento seja de forma correta, a qual será apresentada na sequência.

Durante a pesquisa e a execução do presente trabalho, foi feito um estudo sobre diversas bibliotecas já implementadas e disponibilizadas de forma gratuita para todos. É necessário realizar uma sequência de ação, que será apresentada e explicada passo a passo.

Sabendo disso, para iniciar a configuração, é necessário acrescentar a biblioteca presente já na IDE do Arduino. Para ter acesso as bibliotecas, é necessário acessar as preferências do projeto, como ilustrado na Figura 10.

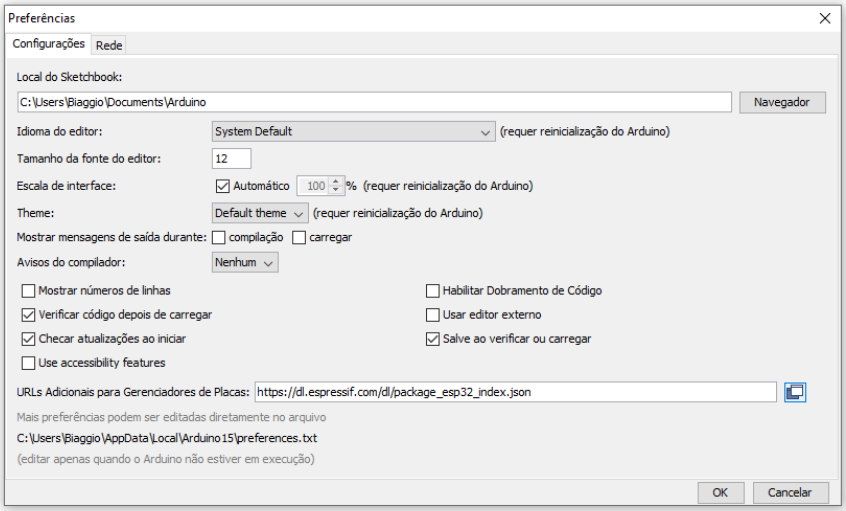
Figura 10 – Caminho para localizar a biblioteca ESP32



Fonte: Autoria própria (2022)

Irá abrir uma janela onde será necessário acrescentar o seguinte diretório "https://dl.espressif.com/dl/package_esp32_index.json" para que seja possível realizar a inserção da biblioteca. Tal processo é demonstrado na Figura 11.

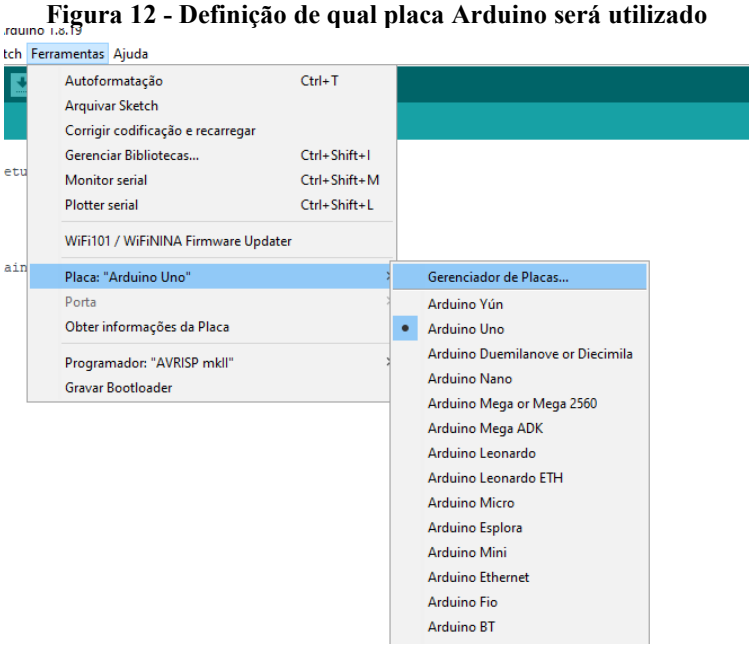
Figura 11 - Inserção do diretório para acrescentar a biblioteca do ESP32



Fonte: Autoria própria (2022)

Após isso, já é necessário realizar a instalação da biblioteca. Como existem vários tipos diferentes de *Arduino* e cada um contém as suas configurações pré-instaladas, é indispensável informar ao programa qual é o modelo que está sendo usado. Como no caso do deste trabalho, foi realizado todo o processo usando as configurações pré-definidas das placas como *Arduino Uno*. Pode ser visto tal passo na Figura 12.

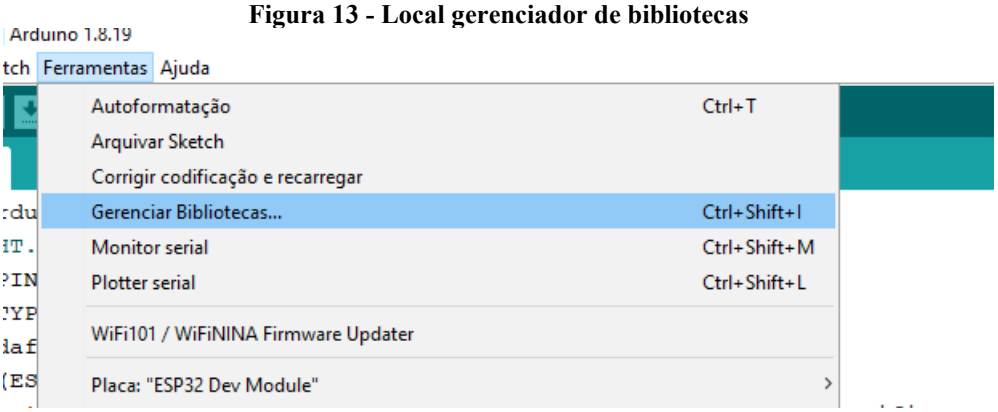
Figura 12 - Definição de qual placa Arduino será utilizado



Fonte: Autoria própria (2022)

Já definido qual modelo de placa será usado, pode-se incluir a biblioteca do ESP32. Para tal, será necessário abrir o gerenciador de bibliotecas, onde pode ser visto o caminho na Figura 13.

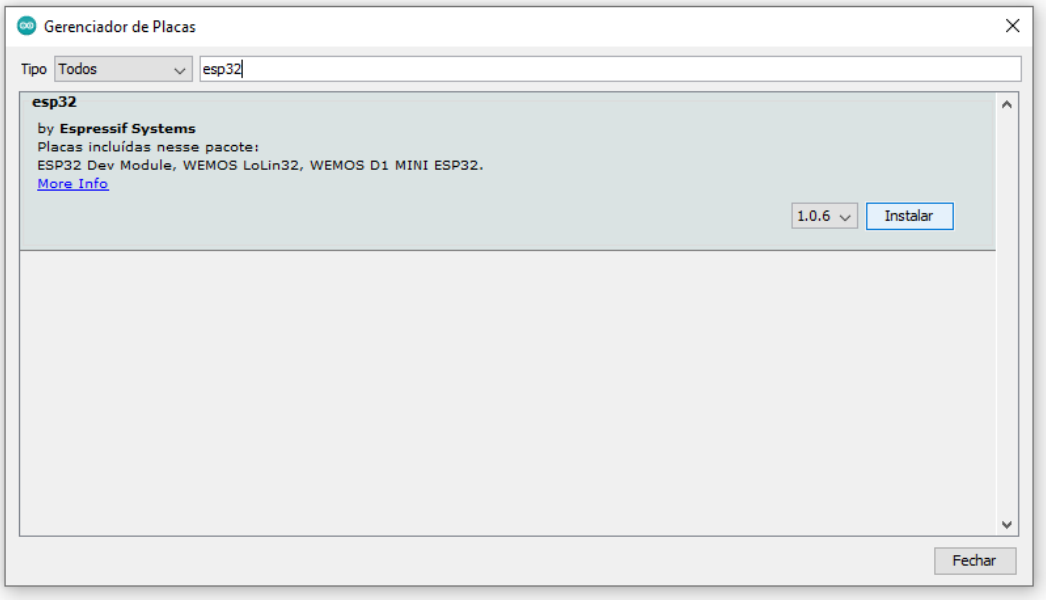
Figura 13 - Local gerenciador de bibliotecas



Fonte: Autoria própria (2022)

Após abrir o gerenciador de bibliotecas, iniciar uma pesquisa no campo reservado para tal e buscar pela palavra “esp32”. Após isso irá ter apenas uma biblioteca, o qual é necessário clicar em instalar e esperar até que o programe instale a biblioteca em sua máquina. Todo o processo explicado pode ser visto na Figura 14.

Figura 14 - Gerenciador de bibliotecas

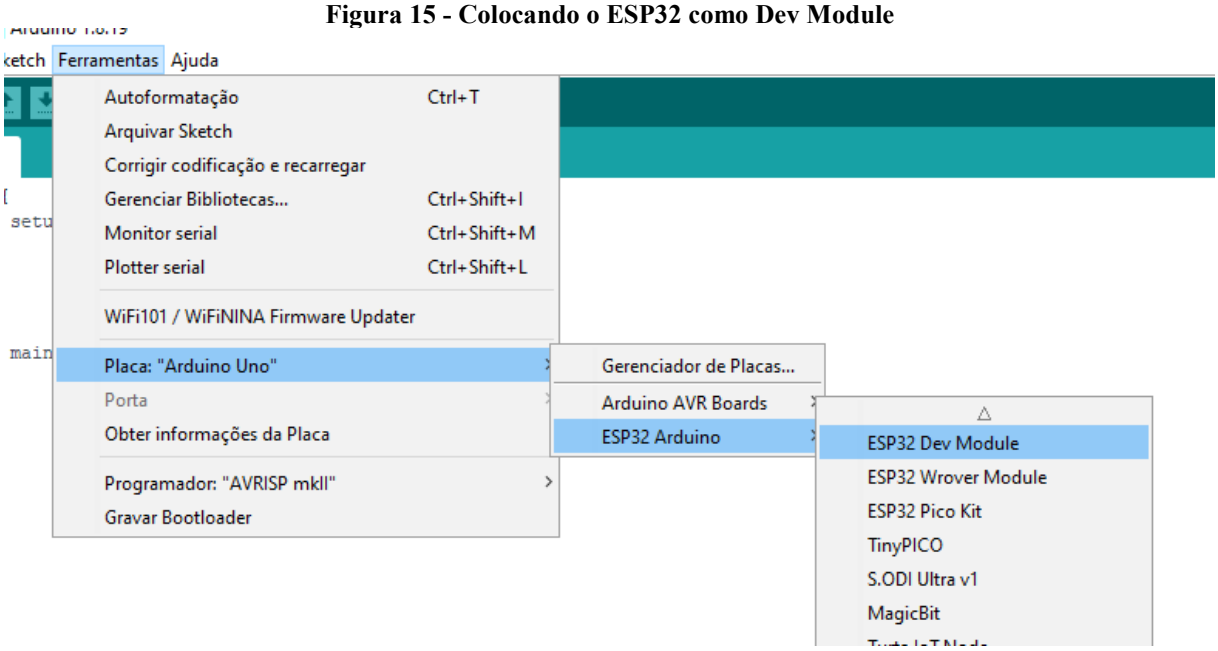


Fonte: Autoria própria (2022)

Para de fato conseguirmos configurar a IDE do Arduino para receber as configurações necessárias para que o ESP32 funcione como o esperado, é necessário

reinicializar a IDE, pois somente assim todas as instalações pendentes de bibliotecas serão instaladas. Após isso, é necessário colocar o ESP32 como o módulo desenvolvedor, como pode ser visto na Figura 15.

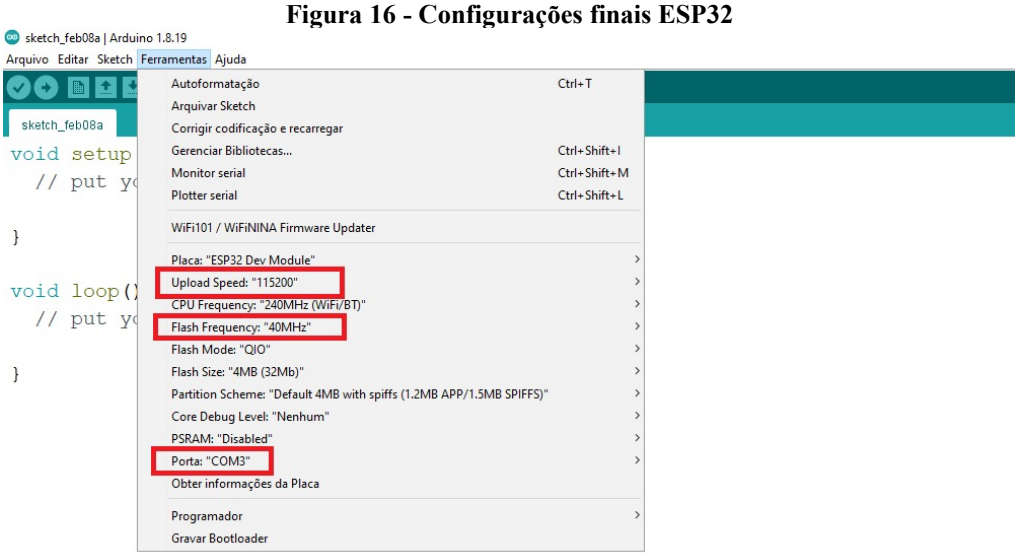
Figura 15 - Colocando o ESP32 como Dev Module



Fonte: Autoria própria (2022)

Após colocar a IDE em módulo de desenvolvedor, as opções, vista na Figura 16, são autorizadas para que possam ser alteradas.

Figura 16 - Configurações finais ESP32



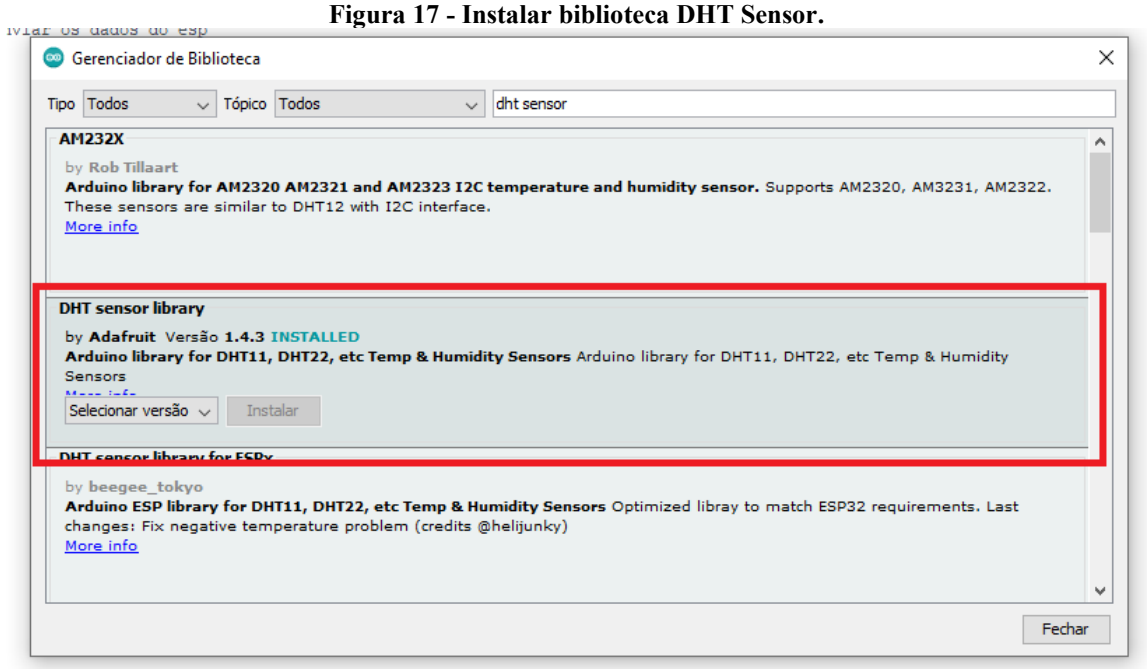
Fonte: Autoria própria (2022)

Para que o ESP32 funcione como esperado, as configurações padrão não serão totalmente utilizadas, tendo em vista que é necessário adequar os dados para cada aplicação. Tendo isso em vista, como visto na Figura 16, os dados que serão modificados são *Upload Speed, Flash Frequency* e a Porta. Para todos os dados alterados, apenas a Porta será diferente para cada caso, pois essa configuração mostra qual porta USB está sendo utilizada para a gravação do código no ESP32.

3.1.2 Instalar bibliotecas

Usando o *Library Manager* da IDE do *Arduino*, é possível acessar o menu "Sketch > Incluir Biblioteca > Gerenciar Bibliotecas". Depois disso é só buscar por "*Adafruit unified sensor*" e "*DHT sensor library*". Tal biblioteca, pode ser vista na Figura 17.

Figura 17 - Instalar biblioteca DHT Sensor.



Fonte: Autoria própria (2022)

3.1.3 Circuito eletrônico

Para realizar o sensoriamento, se faz necessário planejar um sistema eletrônico. Para isso, foram utilizados os seguintes componentes:

- 1 LDR;
- 2 LEDs;
- 1 DHT11;
- 1 ESP32;
- 2 Resistores 10k;
- 2 Resistores 20k;
- 1 Protoboard.

Para a medição de luminosidade foi utilizado o sensor LDR (*Light Dependent Resistor*) ou fotorresistor, o qual é um sensor de luz que é utilizado em inúmeras aplicações por ser muito acessível e ter um funcionamento simples. A construção desse sensor é feita a partir do sulfeto de cádmio, que dependendo o nível de luminosidade altera sua resistência. Em ambientes escuros a sua resistência é extremante elevada na casa de milhões de ohm e em exposição direta a luz, sua resistência cai para alguns milhares de ohms (THOMAZINI e ALBUQUERQUE, 2010).

A sua estrutura é composta por pequenas trilhas de um material condutor em sua superfície, feita desse modo para que possa ter uma maior capacidade de corrente e uma sensibilidade mais fiel ao ambiente. O encapsulamento é feito de um material que não impeça a captação da luminosidade, sendo normalmente de um material transparente (THOMAZINI e ALBUQUERQUE, 2010).

A RS (1997) traz em seu datasheet uma tabela que nos mostra a característica elétrica de tal LDR para uma temperatura de 25ºC:

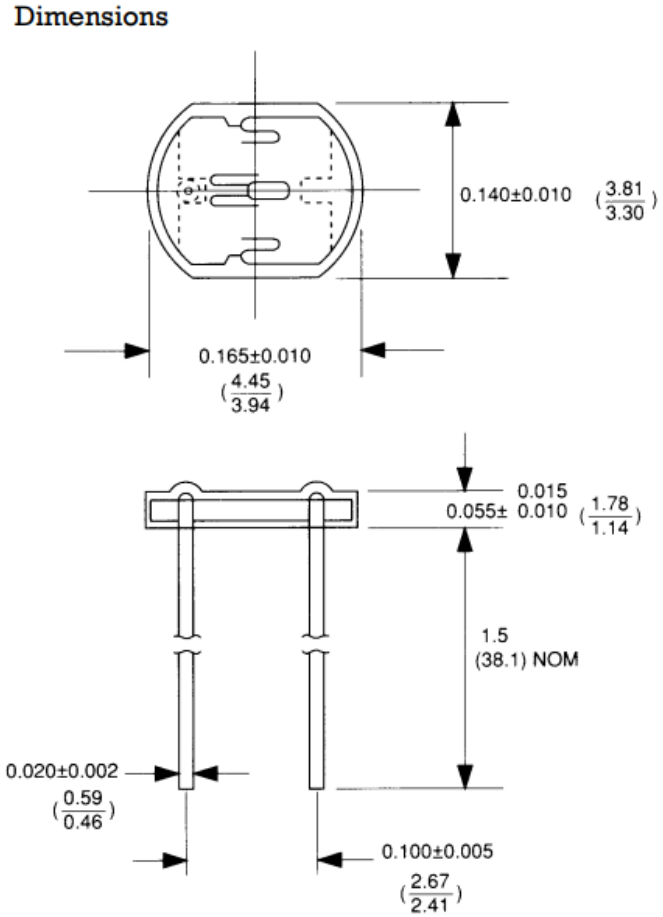
Figura 18 - Tabela característica elétrica LDR

Parâmetro	Conditions	Min.	Typ.	Max.	Units
Cell resistence	10 lux 100 lux	20 -	- 5	100 -	kΩ kΩ
Dark resistence	10 lux after 10 sec	20	-	-	MΩ
Dark capacitance	-	-	550	-	pF
Rise time 1	10 ftc	-	45	-	ms
Fall tume 2	10 ftc	-	55	-	ms

Fonte: RS (1997, p.3)

RS (1997) nos traz na Figura 19, o seu dimensionamento.

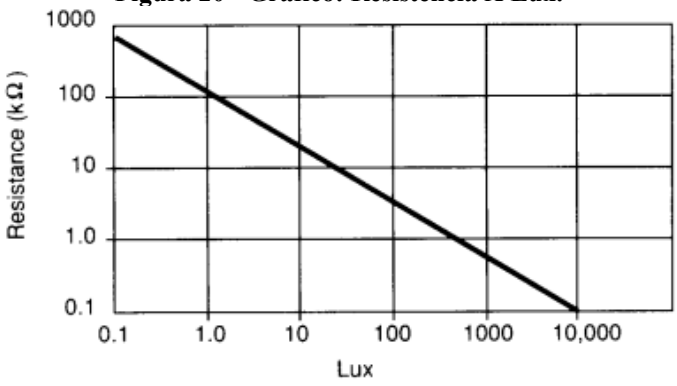
Figura 19 - Dimensão LDR



Fonte: RS (1997, p.3)

Até do seu dimensionamento, RS (2018) também informa a relação entre a resistência X lux, este gráfico pode ser visto na Figura 20.

Figura 20 - Gráfico: Resistência X Lux.

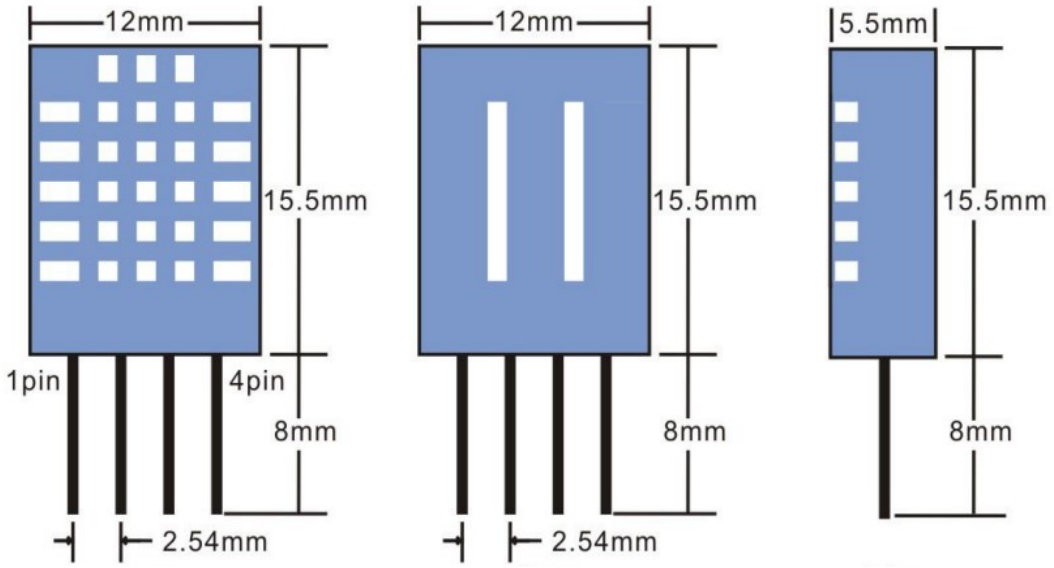


Fonte: RS (1997, p.3).

Para a medição de temperatura e umidade, foi utilizado o sensor DHT11. Segundo *Mouser electronics*, o sensor DHT11 é um sensor que em sua saída nos apresenta um sinal digital que informa o valor de temperatura e umidade. No próprio sensor, existe um sensor resistivo de componentes úmidos e um dispositivo de medição de temperatura NTC.

Para o real conhecimento do dimensionamento do componente, *Mouser electronics* apresenta, na Figura 21, os dados em milímetros.

Figura 21 - Dimensão DHT11



Fonte: MOUSER ELECTRONICS (2011, p.2)

Além disso, *Mouser electronics* revela os parâmetros do funcionamento do sensor.

- Umidade relativa

Resolução: 16 bits

Repetibilidade: ± 1 % RH

Precisão: A 25 °C ± 5 % RH

Intercambialidade: totalmente intercambiável

Tempo de resposta: 1/e (63 %) de 25 °C 6 s

1 m/s ar 6 s

Histerese: < ± 0,3 % RH

Estabilidade a longo prazo: < ± 0,5 % RH / ano em

- Temperatura

Resolução: 16 bits

Repetibilidade: ± 0,2 °C

Alcance: A 25 ℃ ± 2℃

Tempo de resposta: 1/e (63%) 10s

- Característica eletrônica

Fonte de alimentação: DC 3,5~5,5V

Corrente de alimentação: medição 0,3mA em espera 60μA

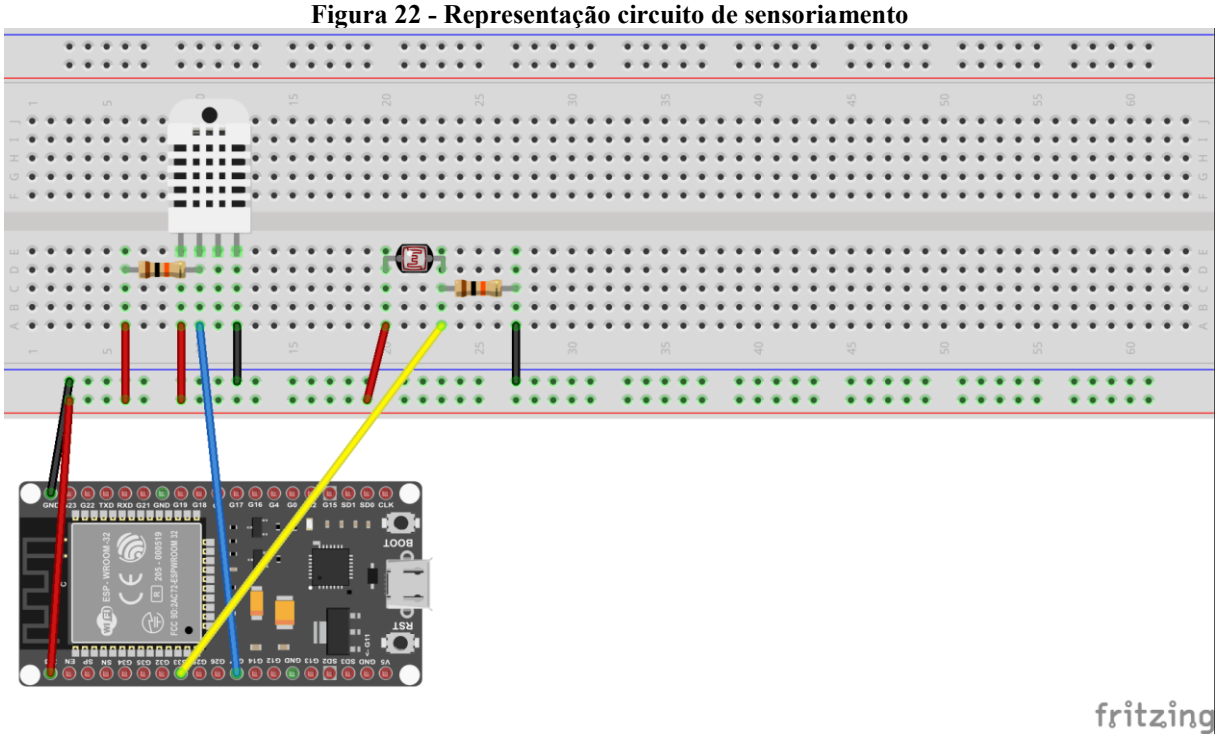
Período de amostragem: mais de 2 segundos

- Descrição dos pinos

1. a fonte de alimentação VDD 3,5 ~ 5,5 V DC
2. dados seriais DATA, um único barramento
3. NC, pino vazio
4. terra GND, a potência negativa

A representação do circuito físico está demonstrada na Figura 22.

Figura 22 - Representação circuito de sensoriamento



Fonte: Autoria própria (2022)

Foi adotado que as trilhas do lado vermelho e azul, que percorrem toda a *protoboard*, são o VCC e o GND, respectivamente, do circuito.

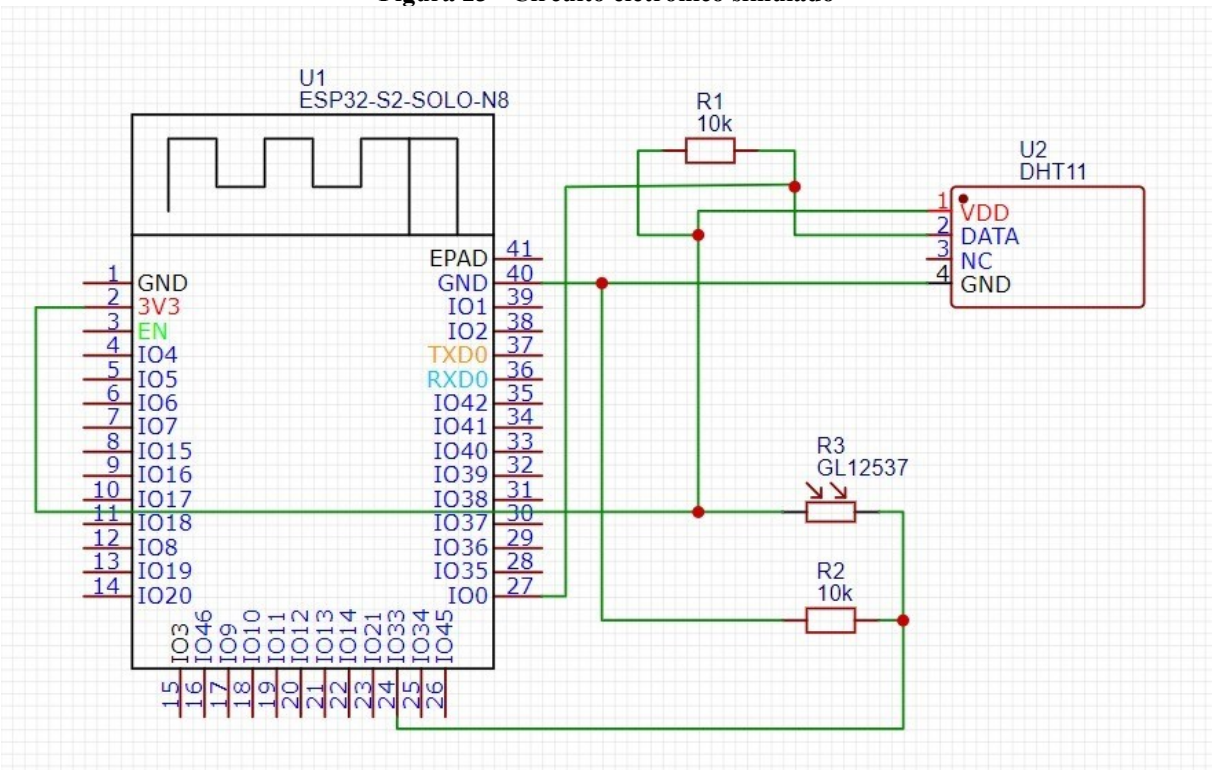
Para o funcionamento correto, é necessário que conecte o GND do ESP32 na trilha que foi atribuída como GND do circuito. Realizou-se o mesmo processo com o VCC, mas conectando-o a trilha correspondendo a ele na *protoboard*.

Nessa etapa foi necessário conectar um resistor de 10k no VCC e no pino 2 do DHT11. Juntamente, foi conectado o pino 2 do DHT11 ao GPIO27 do *ESP32*. Também foi conectado o pino 1, do DHT11, ao VCC e o pino 4 ao GND.

Com o DHT11 devidamente conectado, será conectado o LDR ao sistema. Um dos seus pinos será ligado ao VCC e o outro pino a um resistor de 10k. Também foi necessário ligá-los ao GPIO36 do *ESP32*. Já o segundo terminal do resistor de 10k foi ligado ao GND.

Tal circuito é possível ser visto na Figura 23.

Figura 23 - Circuito eletrônico simulado



Fonte: Autoria própria (2022)

Para uma melhor visualização dos pinos, a Figura 24 descreve onde todos os elementos estão conectados no sistema.

Figura 24 - Pinos conectados.

PIN	GPIO	SENSOR
D27	GPIO27	DHT11

D33	GPIO33	LDR
GND	GND	-
3V3	VIN	-

Fonte: Autoria própria (2022)

3.1.4 Leitura de dados

Para realização da leitura de dados, foram utilizadas funções específicas da biblioteca do DHT. Com isso, a primeira função utilizada foi a “DHT dht(DHTPIN, DHTTYPE)”. Esta função está criando um objeto do tipo DHT com dois parâmetros, os quais são o DHTPIN e DHTTYPE, em que o DHTPIN é um parâmetro que informa o pino em que o sensor está conectado e o DHTTYPE é um parâmetro que informa o tipo do sensor.

Com o objeto criado, é utilizado a função “dht.begin()” que inicializa o objeto “dht” e termina realizando um chamado para a instância DHT.

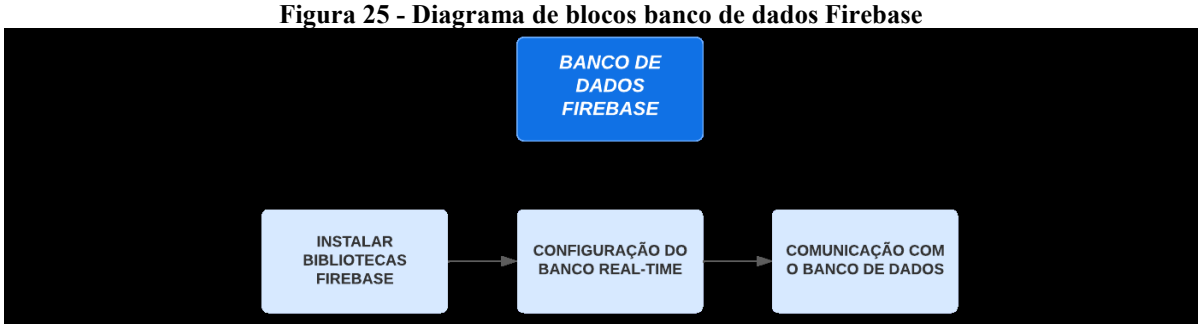
Na sequência, foi realizado a leitura e armazenamento dos valores mensurados pelo sensor. As funções utilizadas são “dht.readHumidity()” e “dht.readTemperature()”. Essas são responsáveis para receber os dados da umidade e temperatura, respectivamente.

A função “analogRead(ldr)” faz um papel semelhante as anteriores, porém ela recebe o valor mensurado do LDR.

3.2 Banco de dados Firebase

Para ter uma visão mais completa de todo o processo, foi realizado um diagrama de blocos com todo o sistema. Tal diagrama pode ser visto na Figura 25.

Figura 25 - Diagrama de blocos banco de dados Firebase



Fonte: Autoria própria (2022)

Tendo em vista todo o processo, será explicado cada processo individualmente em sequência.

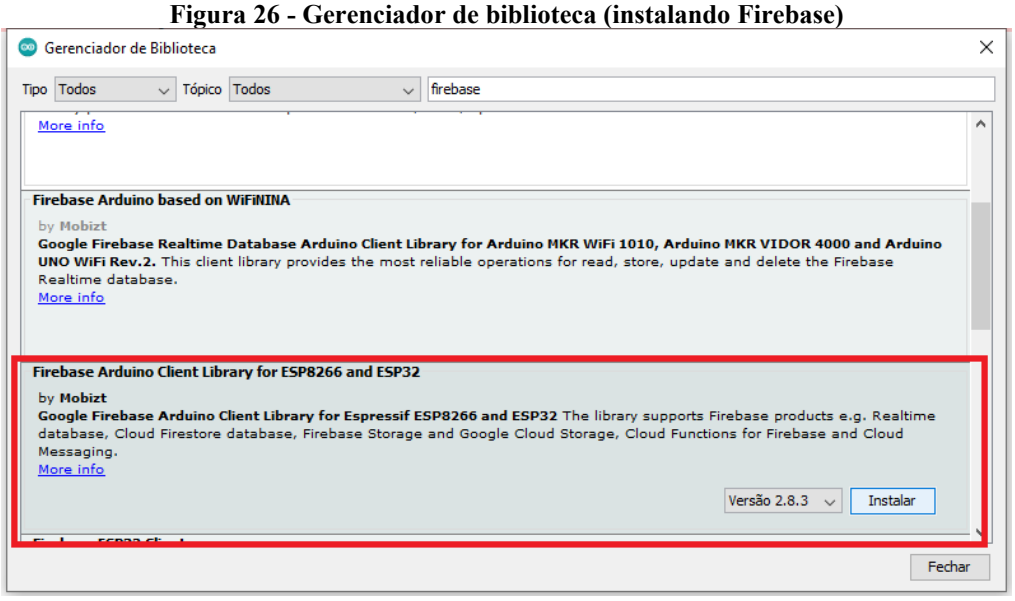
3.2.1 Instalar bibliotecas Firebase

Esta biblioteca permite que faça a comunicação do ESP32 com o *Firebase*. O diferencial dessa biblioteca é a vasta implementação já existente nela, além de ser completa, rápida, segura e além de ter o suporte de poder criar, ler, atualizar e deletar dados diretamente no banco de dados do *Firebase*.

Durante a pesquisa e a execução do presente trabalho, foi feito um estudo sobre diversas bibliotecas já implementadas e disponibilizadas de forma gratuita para todos. Para tal, é necessário realizar uma sequência de ação, que será apresentada e explicada passo a passo.

Será necessário incluir a biblioteca no gerenciador de bibliotecas chamada "*Firebase Arduino Client Libary for ESP8266 and ESP32"*, como mostrado na Figura 26.

Figura 26 - Gerenciador de biblioteca (instalando Firebase)



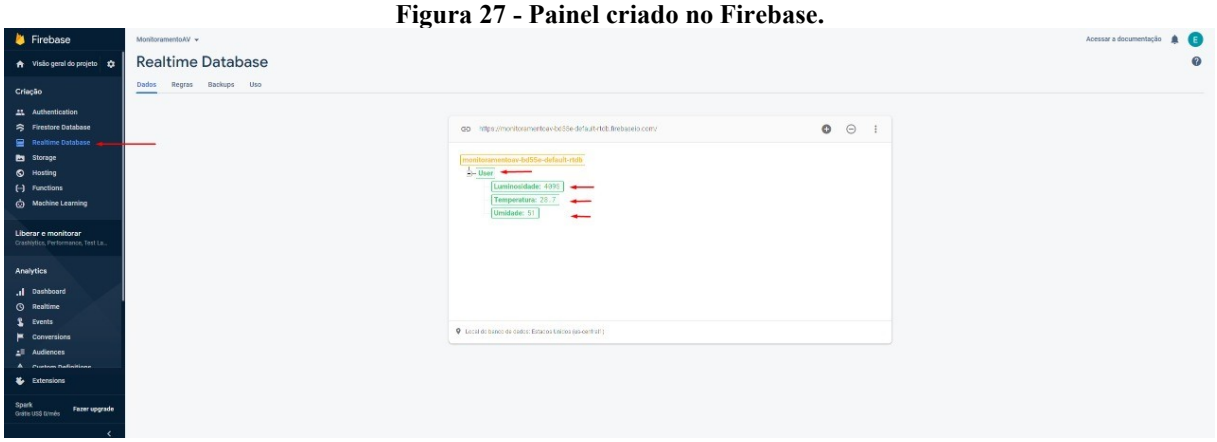
Fonte: Autoria própria (2022)

3.2.2 Configuração do banco *Real-Time*

O banco *Real-Time*, permite que os desenvolvedores criem aplicativos interativos e com uma diversa gama de recursos. A configuração deste banco se faz necessária para que todos os dados coletados pelos sensores possam ser apresentados para o cliente de uma forma correta.

Com isso, é necessário realizar todo o processo para a criação do banco. De início, foi criado um campo onde serão apresentados os dados coletados. Para isso, acessa-se o painel do *Firebase*, o qual foi utilizado o *Realtime Database*, e foi criado um campo *User* que recebe três valores, os quais são temperatura, luminosidade e umidade, como exibido na Figura 27.

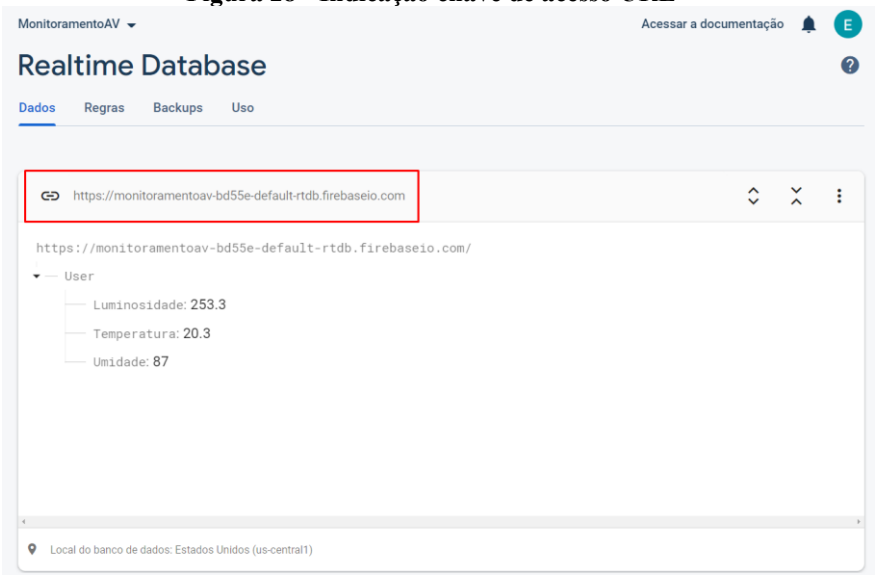
Figura 27 - Painel criado no Firebase.



Fonte: Autoria própria (2022).

Para a comunicação com o banco de dados, se faz necessário a utilização de dois tipos diferentes de chaves de acesso. Uma delas é o URL do banco de dados, que se encontra na Figura 28 demonstrado.

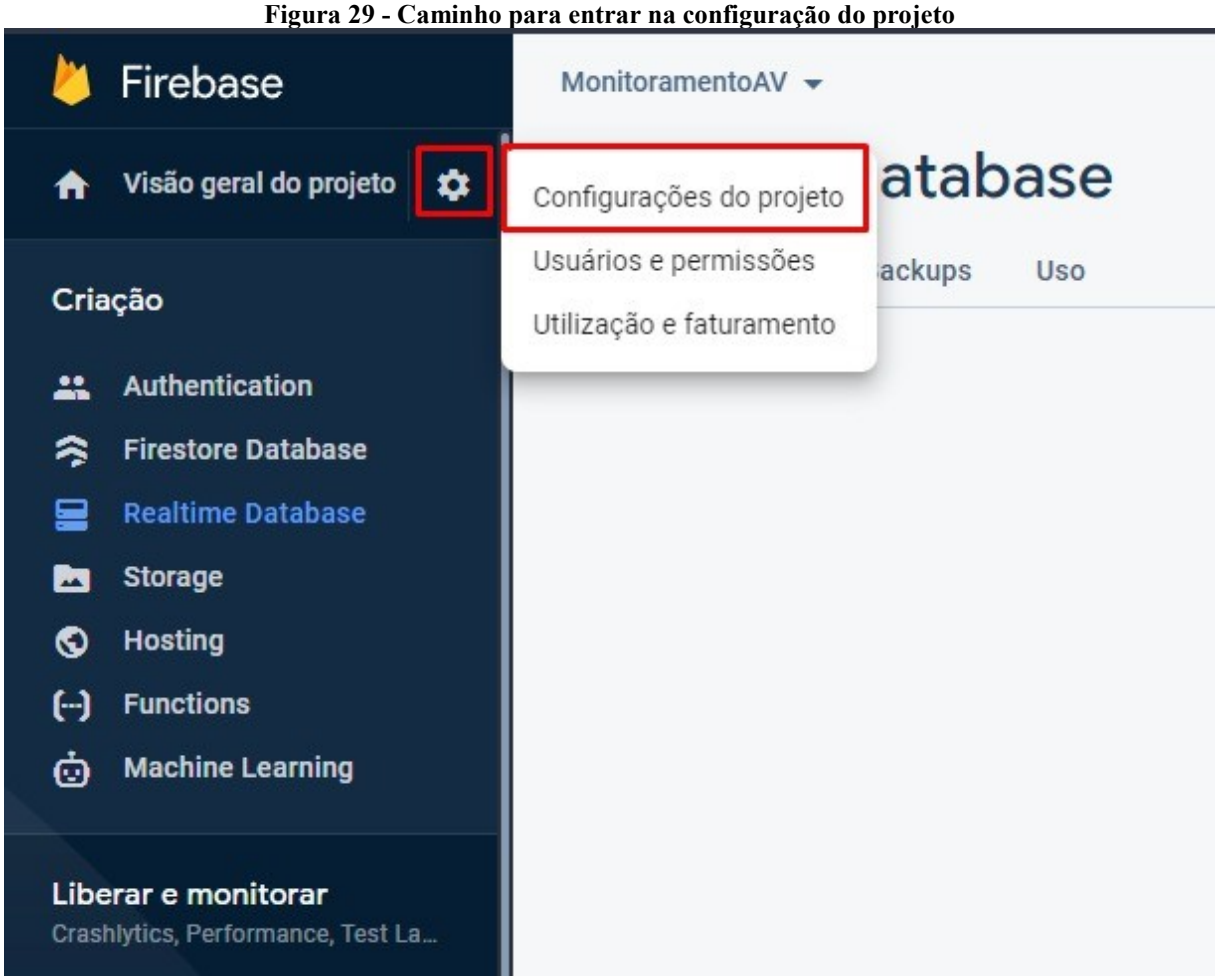
Figura 28 - Indicação chave de acesso URL



Fonte: Autoria própria (2022)

Além da chave URL, é preciso utilizar a chave API. Esta chave de acesso se encontra dentro das configurações do projeto. A Figura 29 demonstra o caminho para entrar na configuração do projeto.

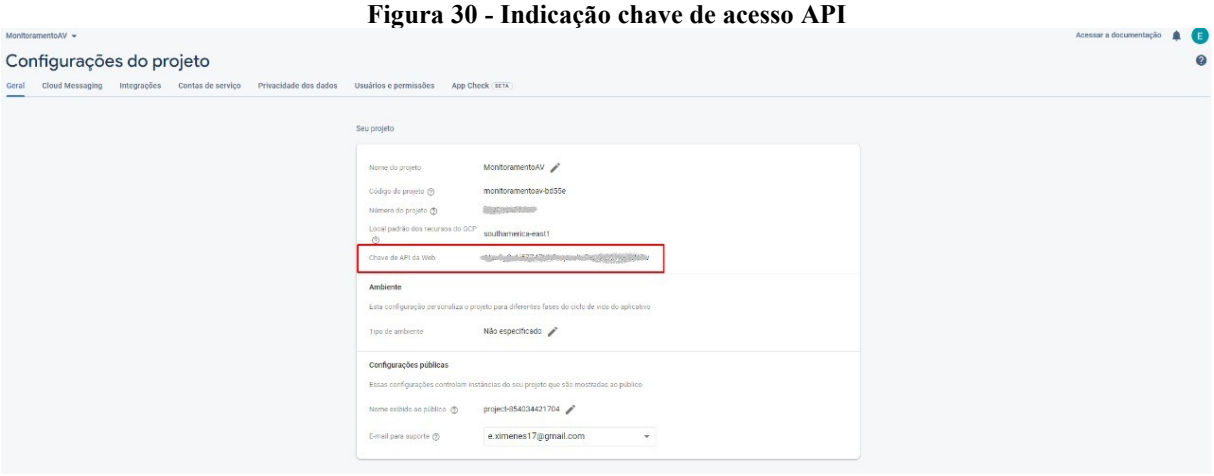
Figura 29 - Caminho para entrar na configuração do projeto



Fonte: Autoria própria (2022)

Ao acessar a configuração do projeto, como mostrado na Figura 30, se encontra a chave API. Após isso, pode ser realizado a comunicação do banco de dados com o projeto desenvolvido.

Figura 30 - Indicação chave de acesso API



Fonte: Autoria própria (2022)

3.2.3 Comunicação com o banco de dados

Ao longo de toda a execução deste trabalho, foram realizadas pesquisas sobre como implementar a comunicação entre o ESP32 e o banco de dados. Dessa forma, foi utilizado a biblioteca específica, dentro dessa biblioteca existe uma grande quantidade de funções já implementadas, onde algumas são necessárias para o funcionamento do código.

A primeira função utilizada é a “FirebaseData fbdo”. A utilização desta se faz necessária para a criação de um objeto chamado “fbdo”. Essa criação é o responsável pela manipulação dos dados no banco de dados.

Na sequência, foi utilizada a função “FirebaseAuth auth”, o qual faz uma autenticação com o banco de dados para que possa ser feita, de forma segura, toda a comunicação entre o ESP32 e o banco de dados.

Com a comunicação concluída, é necessário realizar a configuração dos dados do banco de dados. Para isso, a função “FirebaseConfig config” faz toda essa configuração, além de armazenar os valores da chave API e da URL com as funções “config.api_key” e “config.database_url”, respectivamente.

Após toda a configuração da comunicação com o banco de dados, pode-se iniciar a comunicação. A “Firebase.begin(&config, &auth)” faz exatamente essa tarefa, onde recebe dois parâmetros, o primeiro recebe o resultado final da “FirebaseConfig config” e o segundo recebe o resultado final da “FirebaseAuth auth”.

Para ter certeza de que existe uma comunicação WIFI, foi utilizado a “Firebase.reconnectWiFi(true)”.

Com a comunicação efetuada, pode começar os envios dos dados, e a primeira função utilizada para isso foi “Firebase.RTDB.setFloat(&fbdo, "/User/Luminosidade", *light*)”, esta função é específica para o *RealTime DataBase* (RTDB), possuindo várias funcionalidades utilizando *set's* e *get's* para manipular dados de várias formas. Neste caso foi utilizado o “*getFloat*”, onde irá receber o valor *float* que se encontra na variável light, para o caminho "/User/Luminosidade" no banco de dados. A próxima função, para envio de dados, é a “Firebase.RTDB.getFloat(&fbdo, "/User/Luminosidade")”, esta função é uma função semelhante a “Firebase.RTDB.setFloat(&fbdo, "/User/Luminosidade", *light*)”, porém é utilizado o “*setFloat*”, onde irá enviar um valor *float* que se encontra na variável *light*, para o caminho "/User/Luminosidade" no banco de dados.

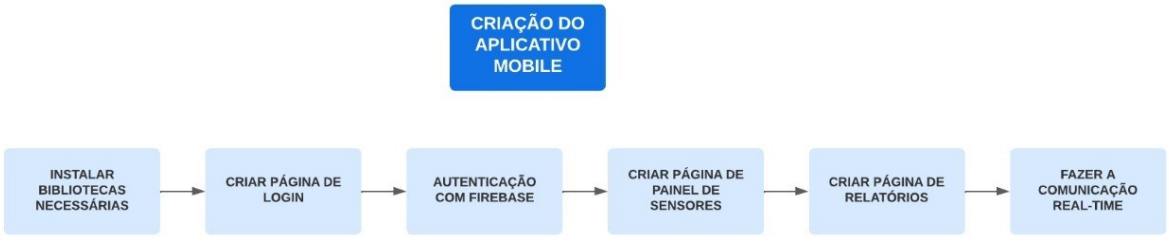
A “fbdo.dataType()” foi utilizada para realizar uma verificação do tipo do dado que está sendo armazenado no objeto “fbdo”.

Na função “fbdo.floatData()”, é realizado um acesso a um dado *float* que está armazenado no objeto “fbdo”.

3.3 Criação do aplicativo mobile

Para ter uma visão mais completa de todo o processo, foi realizado um diagrama de blocos com todo o sistema. Tal diagrama pode ser visto na Figura 31.

Figura 31 - Diagrama do sistema aplicativo mobile



Fonte: Autoria própria (2022)

Tendo em vista todo o processo, será explicado cada processo individualmente em sequência.

3.3.1 Instalar bibliotecas necessárias

As bibliotecas e suas dependências necessárias para execução completa do aplicativo *mobile* foram:

Firebase, que é responsável pelas funções de comunicação com o banco de dados, também é necessário fazer configurações prévias como inserir um arquivo "google-services.json" nas dependências do aplicativo, inserir dados de acesso nas configurações de acordo com a documentação do próprio *Firebase*.

React Navigation que é responsável pela navegação entre telas e enfileiramento de páginas.

React SVG, responsável pela criação de gráficos no aplicativo.

3.3.2 Criar página de login

Na página de "*Login"*, inserimos uma *Logo* para identificação, dois campos de *Input* de texto, que são responsáveis para autenticação de "E-mail" e "Senha", respectivamente, e um botão de "*Login*". Ao clicar no botão, as informações dos campos são enviadas para o banco de dados e após o usuário ser validado ele é redirecionado para a página de "*Home"*.

3.3.3 Autenticação Firebase

A autenticação do usuário no *Firebase* é feita através de uma função "loginFirebase", onde faz uma chamada ao método:

firebase.auth().signInWithEmailAndPassword(email, password)

Essa chamada utiliza os valores dos campos de "E-mail" e "Senha" que foram inseridos nos *Inputs* na tela de "*Login"* e compara com os usuários cadastrados no banco de dados.

3.3.4 Criar página de painel de sensores

Na página "*Home"*, são exibidos os valores dos sensores em tempo real, em que são atualizados em aproximadamente 3 segundos.

A página é composta por 2 botões e 3 campos de exibição de texto, o botão de "Relatório" que é responsável pelo acesso dos gráficos gerados do dia anterior dos dados armazenados e o botão "Sair" que volta para tela de "*Login"*.

Os campos de exibição do painel são dinamicamente alterados de acordo com a leitura dos dados do banco de dados, onde são exibidos dados de Temperatura, Umidade e Luminosidade.

3.3.5 Criar página de relatório

Na página de relatórios, será possível observar três gráficos, os quais são temperatura, umidade e luminosidade. Esses dados estão sendo adquiridos do banco de dados em um monitoramento de duas em duas horas, em doze amostras totalizando vinte e quatro horas de monitoramento.

Os valores obtidos nos gráficos são dinâmicos e são alterados conforme são enviados os dados para o banco de dados.

3.3.6 Comunicação tempo real

Para realizar a leitura de dados no banco de dados, foi criada uma função “Sensor”, em que acessa o banco de dados com as configurações do *Firebase* previamente realizadas e com o caminho de referência “*User*” em que foi criado no *Firebase Realtime*. Ao acessar recebemos um *Array* de valores que são salvos em uma variável “sensorValues: {“Temperatura”: valor, “Umidade”: valor, “Luminosidade”: valor}”, com isso é possível filtrar e separar os valores para serem mostrados individualmente utilizando a chamada “sensorValues.Temperatura”, “sensorValues.Umidade” e “sensorValues.Luminosidade” e esses dados foram exibidos através de um componente de texto na tela.

3.4 Testes e ensaios

Para a validação do projeto final, foi feita uma sequência de testes com o intuito de verificar a eficiência do projeto.

Para a validação do sensor de temperatura foi realizada uma comparação do valor atual e aproximando uma fonte de calor, verificando se há alteração dos valores adquiridos.

Utilizando a mesma fonte de calor, fui utilizado para realizar uma diminuição da porcentagem da umidade no ambiente e foi verificado se há alteração dos valores adquiridos.

Após esse processo, foi retirado a fonte de calor e verificado se há alteração nos valores adquiridos.

Para a validação do sensor de luminosidade foi realizado uma comparação do valor atual e aproximando uma fonte de luz, verificando se há alteração dos valores adquiridos. E de forma similar foi realizada uma diminuição da luminosidade do ambiente, verificando se há alteração dos valores adquiridos. Será realizada uma comparação dos dados obtidos da porta serial do ESP32 com os dados armazenados do banco de dados e com os dados exibidos pelo aplicativo.

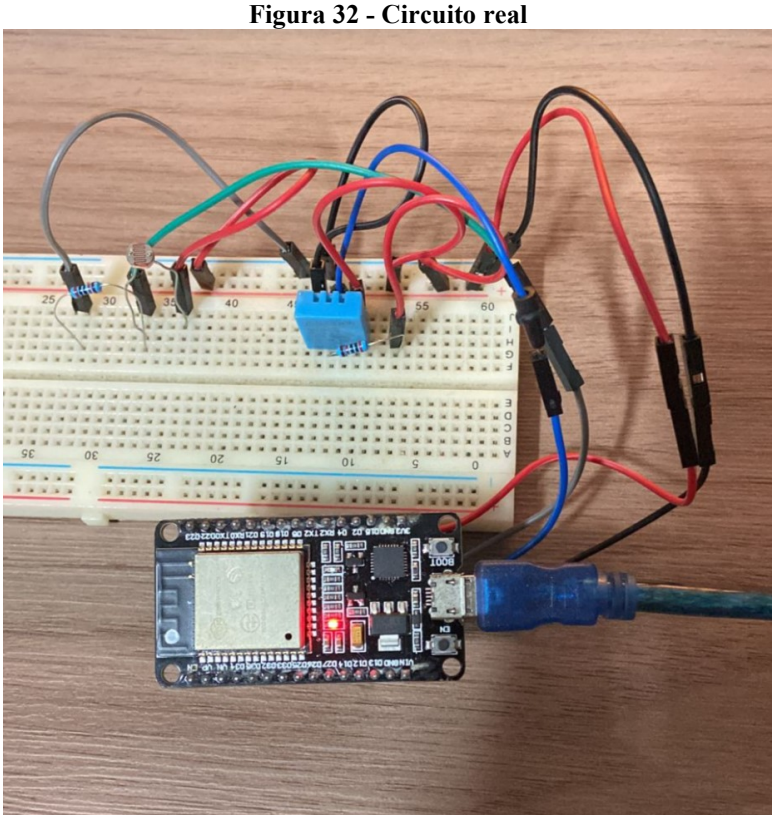
4 RESULTADOS E DISCUSSÕES

Nesse capítulo serão apresentados os resultados obtidos pela execução da metodologia, assim como, uma análise do protótipo afim de verificar se os objetivos foram alcançados e possíveis melhorias.

4.1 Sensoriamento

O circuito eletrônico foi montado em uma *protoboard* utilizando *jumpers* para realizar as ligações, como visto na Figura 32. A alimentação foi realizada utilizando a saída 3,3 V do próprio ESP32, o qual está sendo alimentado via USB.

Figura 32 - Circuito real

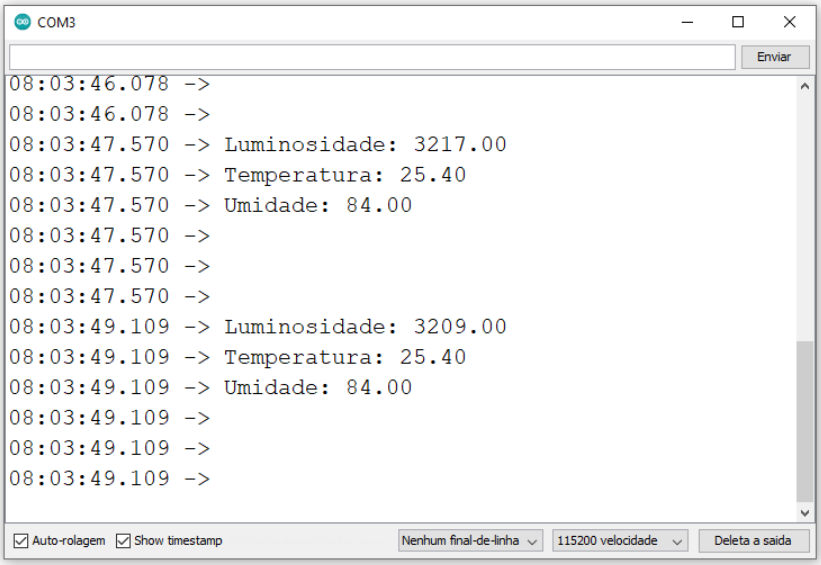


Fonte: Autoria própria (2022)

Para identificar os valores que são lidos pelo ESP32 dentro da IDE, é utilizado um monitor serial, onde é possível ler os dados dos sensores e todas as informações que foram previamente configuradas para serem exibidas, como os status de autenticação do WIFI, *Firebase* e os dados dos sensores.

Na Figura 33 são apresentados os dados na saída do ESP32 através do monitor serial da IDE, onde são observados os dados de leitura dos sensores.

Figura 33 - Monitor serial com os dados do sensor

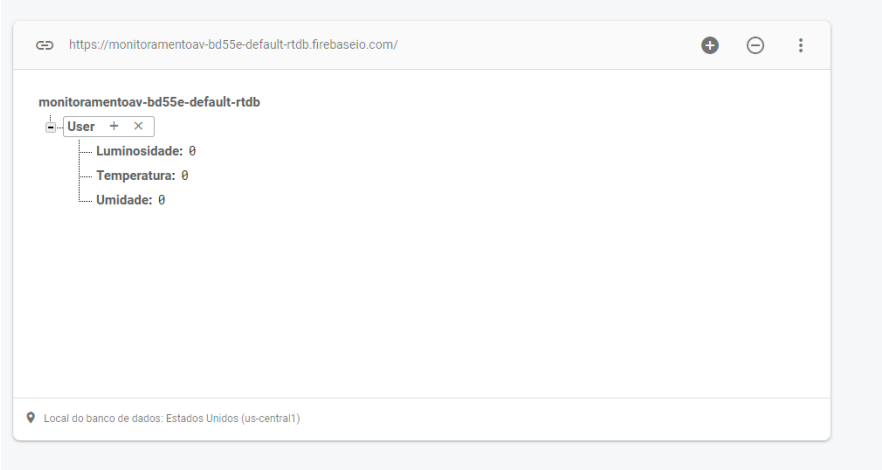


Fonte: Autoria própria (2022)

4.2 Banco de dados

Com os dados sendo recebidos corretamente no monitor serial da IDE e o banco de dados *Firebase* criado e configurado corretamente, foi realizada a comunicação com o *Firebase* e os dados foram enviados em tempo real, na Figura 34 pode-se verificar que inicialmente os dados do banco de dados estão zerados.

Figura 34 – Dados iniciais no banco de dados



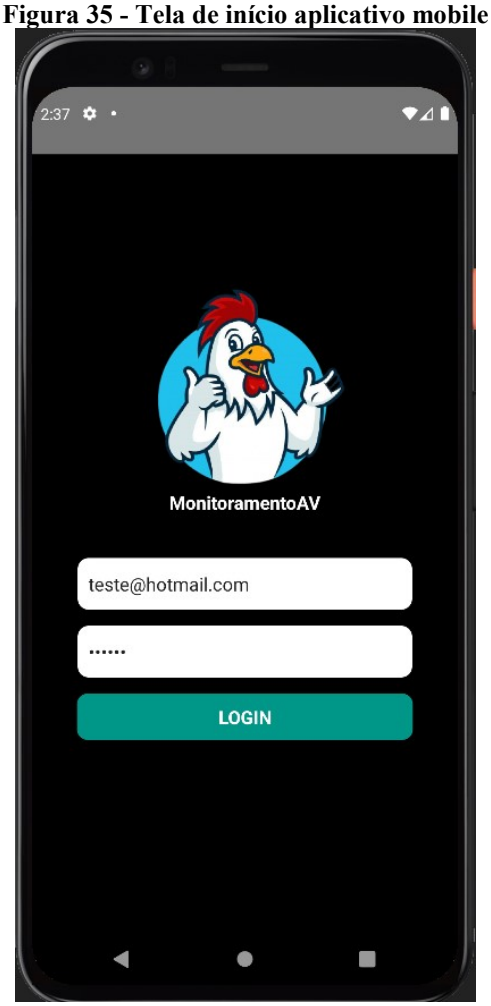
Fonte: Autoria própria (2022)

Após o sistema ser ligado, os dados são atribuídos em suas respectivas variáveis e enviados para o banco de dados onde são armazenados individualmente, tendo seu valor substituído para cada alteração.

4.3 Aplicativo mobile

No aplicativo *mobile*, inicialmente é mostrada uma tela de "*Login"* para realizar uma autenticação no sistema via banco de dados, como visto na Figura 35, esses dados de autenticação são previamente configurados no banco de dados.

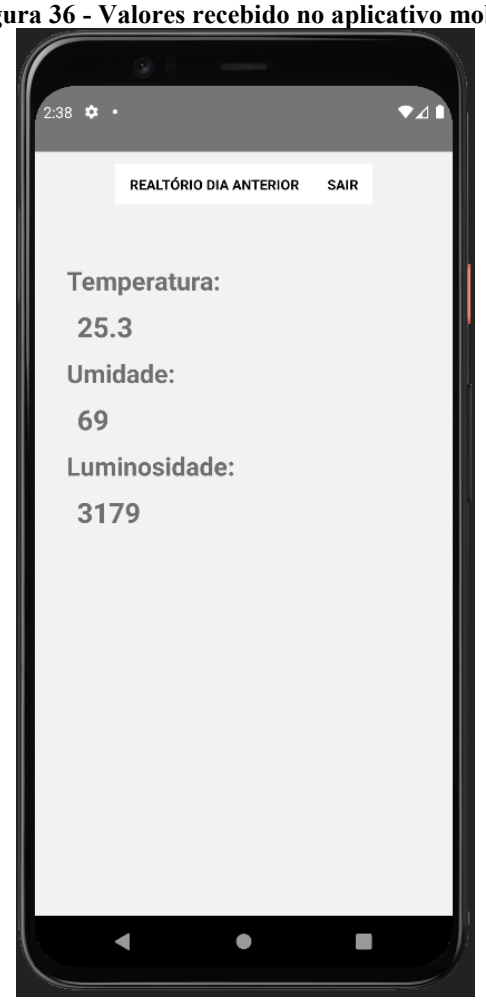
Figura 35 - Tela de início aplicativo mobile



Fonte: Autoria própria (2022)

Após realizar o "*Login"* corretamente, redireciona-se para a página "*Home"* do aplicativo, onde são exibidos painéis contendo os dados dos sensores em tempo real que são recebidos do banco de dados, como visto na Figura 36.

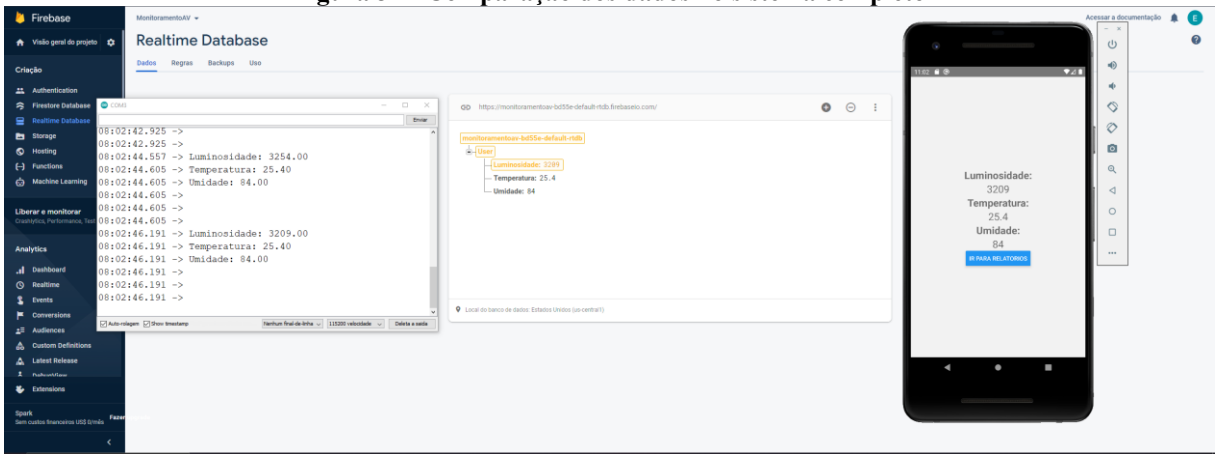
Figura 36 - Valores recebido no aplicativo mobile



Fonte: Autoria própria (2022)

Para verificar se os dados estão sendo enviados corretamente para o aplicativo, foi comparado em tempo real com os dados da IDE e do banco de dados, como mostrado na Figura 37.

Figura 37 - Comparação dos dados no sistema completo

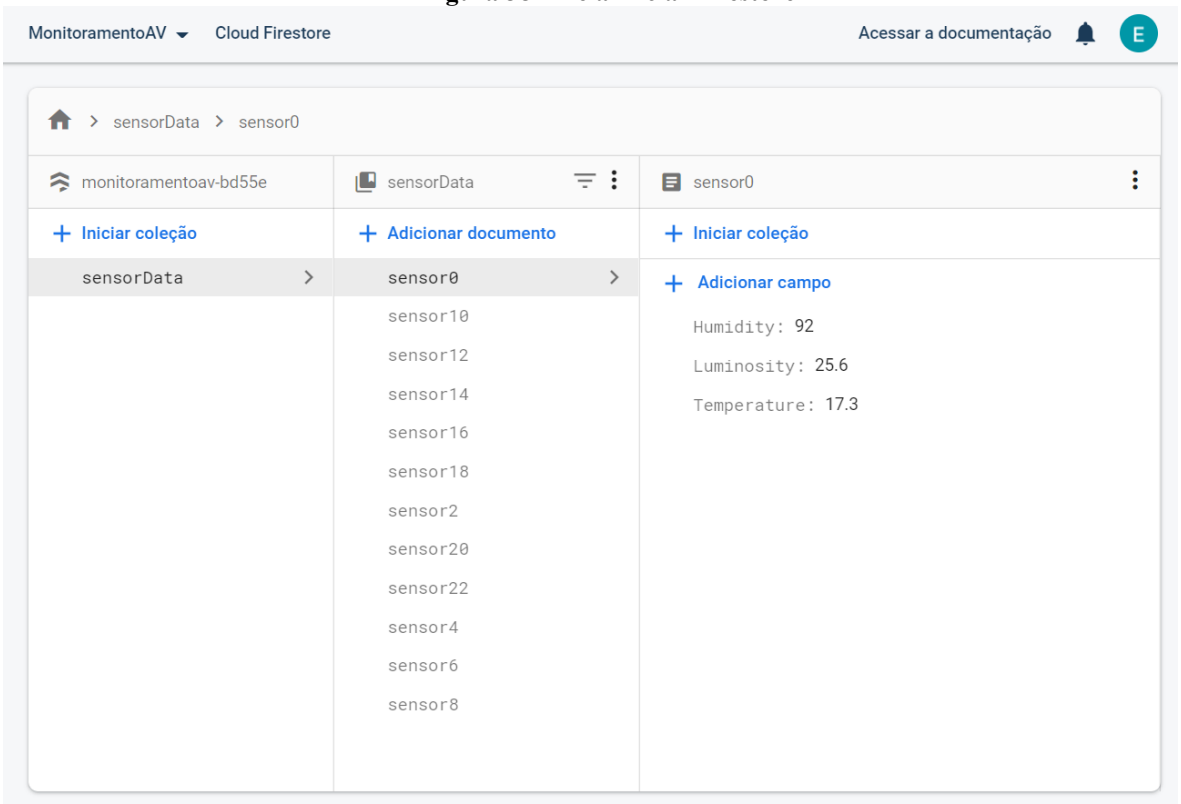


Fonte: Autoria própria (2022)

4.4 Relatórios

Como os dados do banco *Realtime* são perdidos, foi realizado também uma inserção de dados no *Firebase Firestore*, onde existem documentos chamados de coleções e dentro desses documentos podem ser armazenados valores dos sensores como na Figura 38.

Figura 38 – Tela inicial Firestore



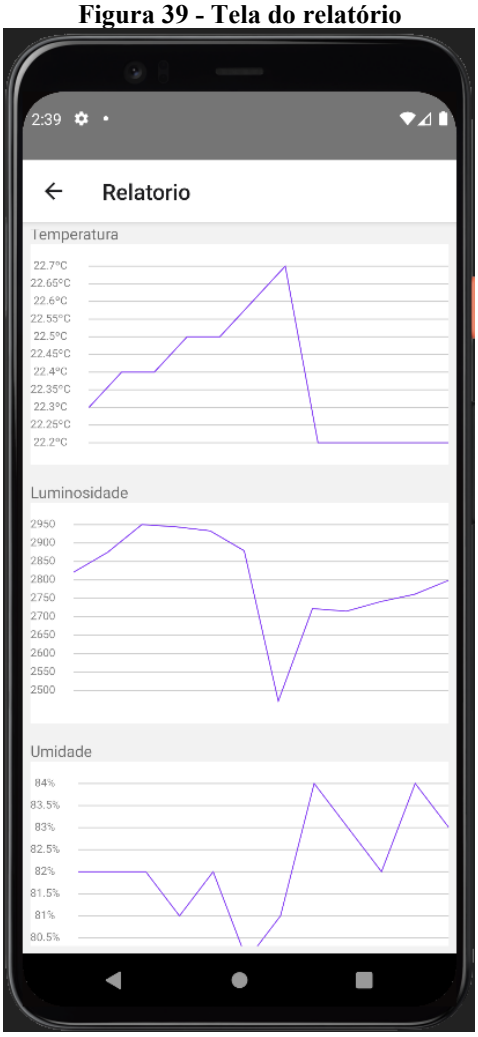
Fonte: Autoria própria (2022)

Para um monitoramento diário completo, foram armazenadas doze amostras coletadas a cada duas horas. Essas amostras são lidas e armazenadas em um *Array* onde o aplicativo utiliza esses dados para criação dos gráficos com ajuda da biblioteca “React SVG”.

Os gráficos mostram o range mínimo e máximo de acordo com os dados coletados, ou seja, se a menor temperatura coletada for de 22ºC, o menor valor se manterá nessa faixa, evitando que o gráfico fique de difícil leitura caso os valores sejam muito divergentes nesse período.

Também foi adicionada uma rolagem da página para baixo e para cima com o intuito de ter uma melhor visualização dos gráficos, possibilitando a adição de mais sensores e gráficos. Na Figura 39 é possível ver o resultado final dos gráficos de um dia completo.

Figura 39 - Tela do relatório



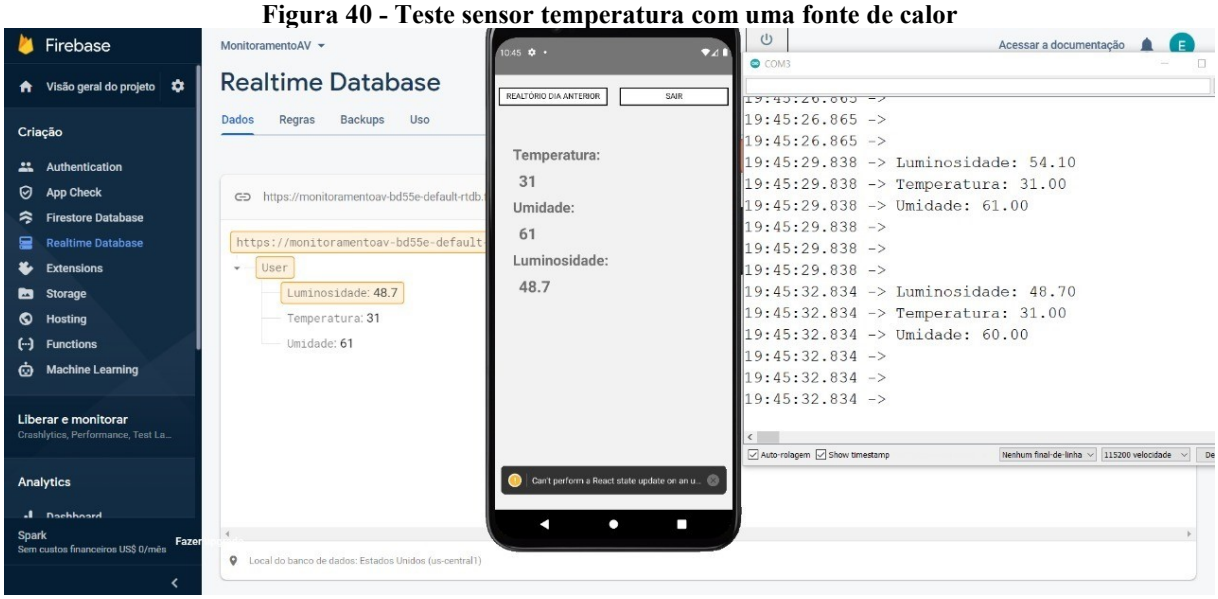
Fonte: Autoria própria (2022)

4.5 Testes e ensaios

Primeiramente realizou-se um teste para validar os sensores de temperatura e umidade o qual aproximou-se uma fonte de calor perto dos sensores.

Após alguns instantes o valor de temperatura do sistema ficou em torno de 31ºC e o valor de umidade do sistema aproximadamente em 61%. Tais mudanças podem ser vista na Figura 40.

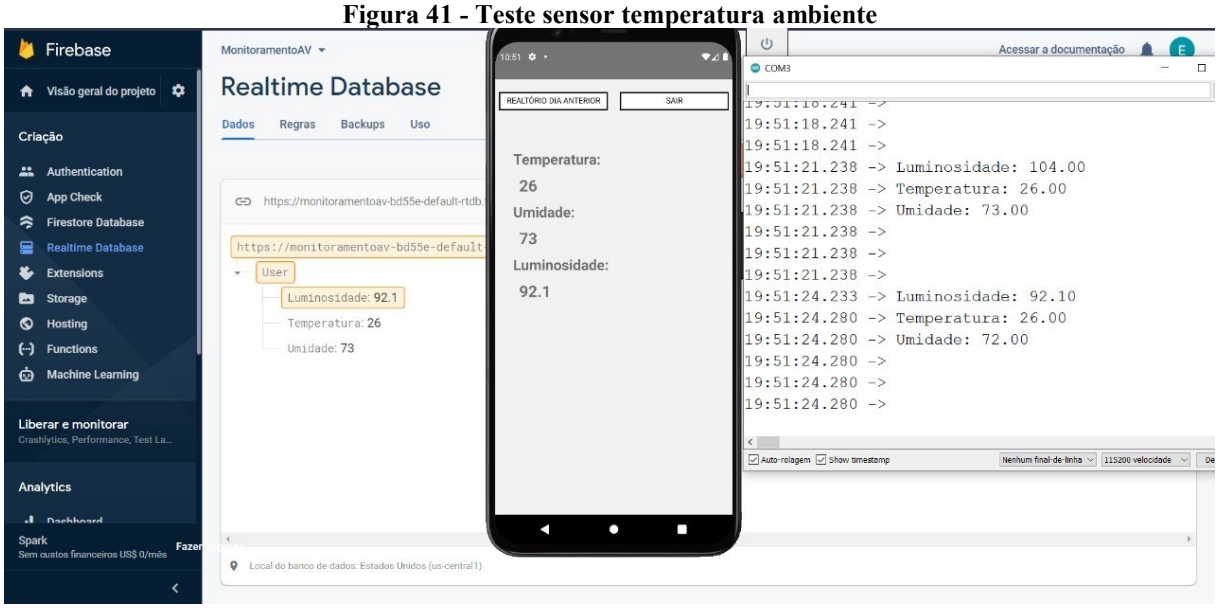
Figura 40 - Teste sensor temperatura com uma fonte de calor



Fonte: Autoria própria (2022)

Após a fonte de calor ser retirada, em alguns instantes os valores normalizaram para os valores de temperatura e umidade ambiente. Tais valores podem ser vistos na Figura 41.

Figura 41 - Teste sensor temperatura ambiente

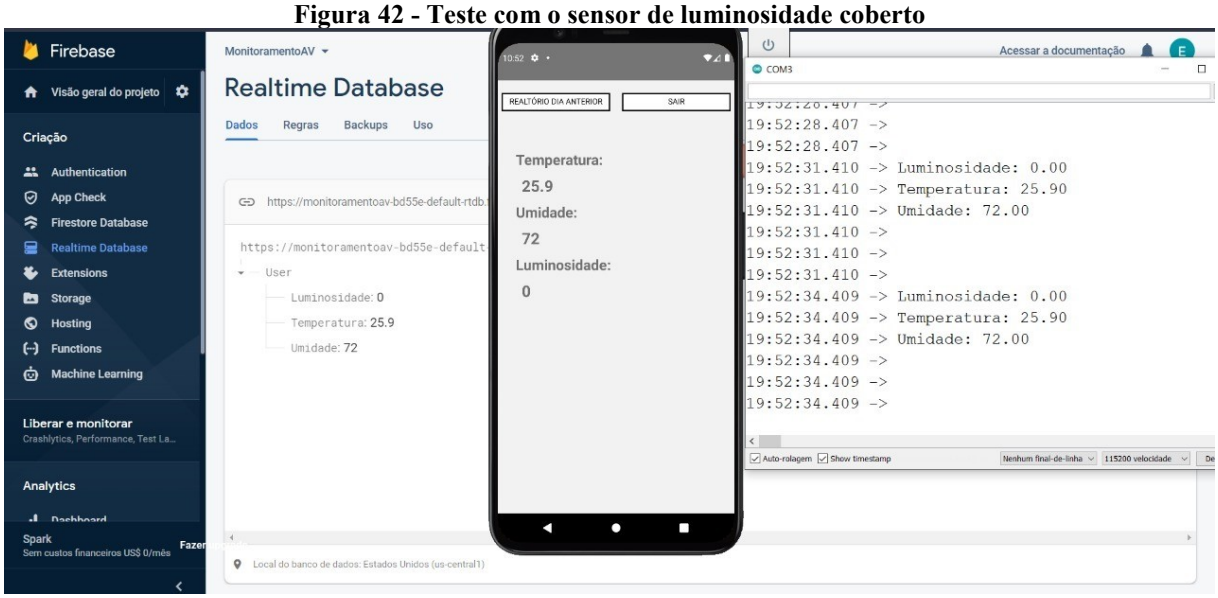


Fonte: Autoria própria (2022)

Com isso, foi possível verificar que os valores de umidade e temperatura estão tendo as alterações esperadas.

Para a validação do sensor de luminosidade foram realizados os seguintes testes. Inicialmente, o sensor foi totalmente coberto com um pano, obstruindo totalmente a passagem de luminosidade, com isso, foi verificado o valor do sensor ficou zerado, como na Figura 42.

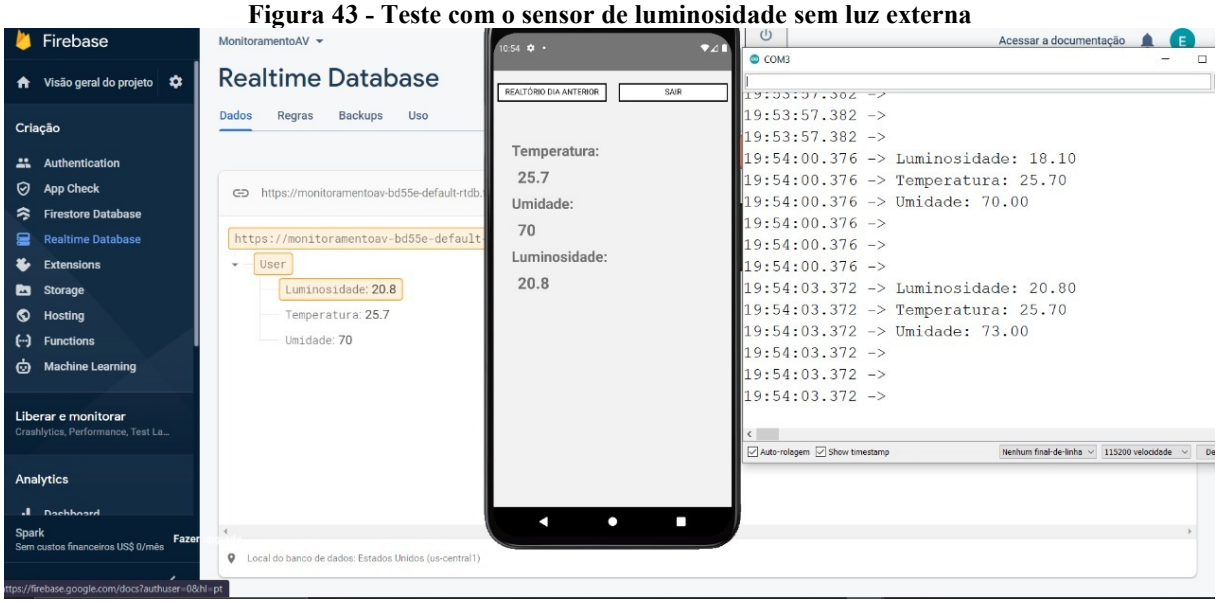
Figura 42 - Teste com o sensor de luminosidade coberto



Fonte: Autoria própria (2022)

Após isso, retirou-se o pano e apagaram-se qualquer fonte de luz externa presente no quarto. O valor obtido foi de 20,8 LUX. Esse processo, pode ser visto na Figura 43.

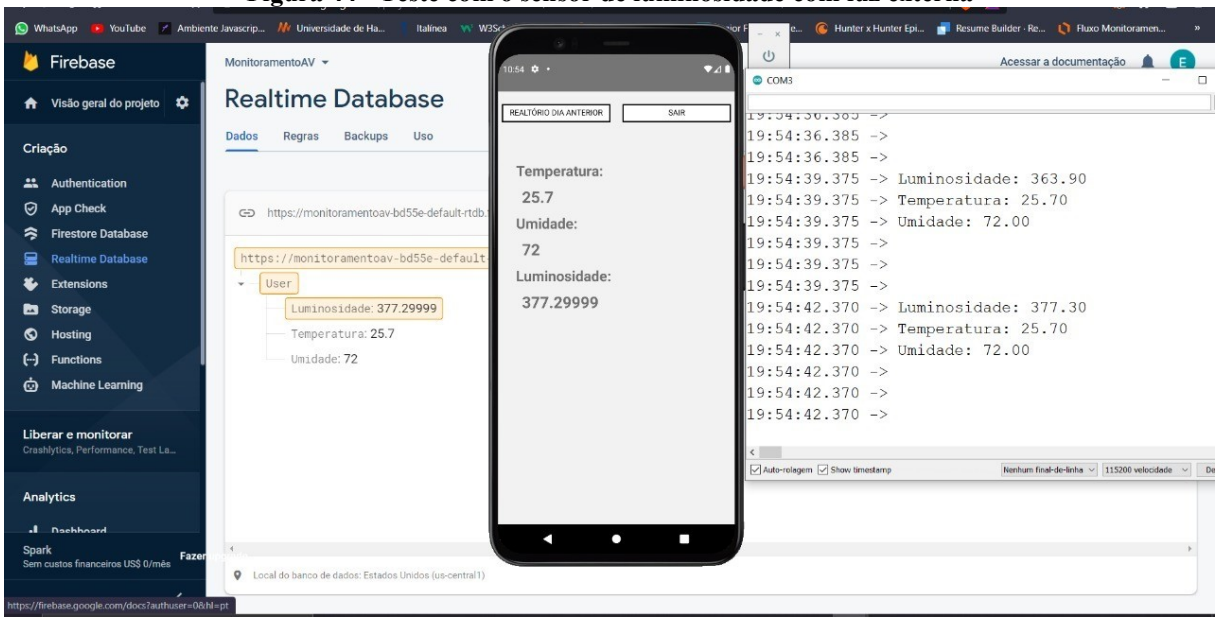
Figura 43 - Teste com o sensor de luminosidade sem luz externa



Fonte: Autoria própria (2022)

E por fim, o último teste foi realizado com as fontes de luzes externa totalmente ligadas e sem obstrução ao sensor. A Figura 44 mostra o sistema nesse estado.

Figura 44 - Teste com o sensor de luminosidade com luz externa



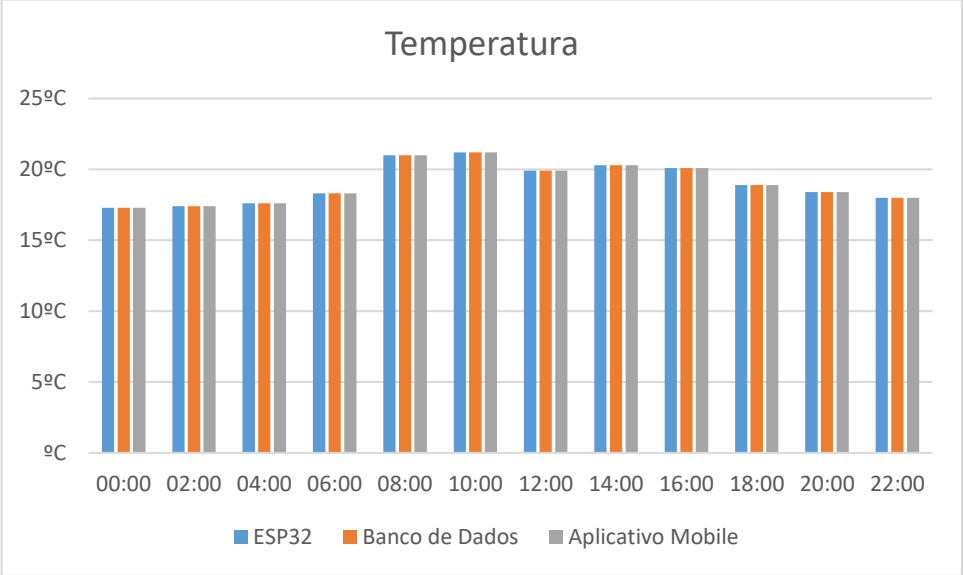
Fonte: Autoria própria (2022)

Para a validação do fluxo de dados do sistema completo e para a confirmação de que os valores estão iguais em todos os processos, realizou-se coleta de dados de forma individual em cada parte do sistema, as quais são a porta serial do ESP32, no banco de dados e no aplicativo *mobile*. Essas amostras foram coletadas a cada duas horas, totalizando um fluxo de vinte e quatro horas.

Tais amostras foram plotados em gráficos separando-os, respectivamente, em temperatura na Figura 45, umidade na Figura 46 e luminosidade na Figura 47.

As amostras foram coletadas no dia 07/05/2022, as medidas de temperatura durante as vinte e quatro horas não tiveram grandes mudanças, mantendo sua faixa entre 17ºC ~ 21ºC como demonstrado na Figura 45.

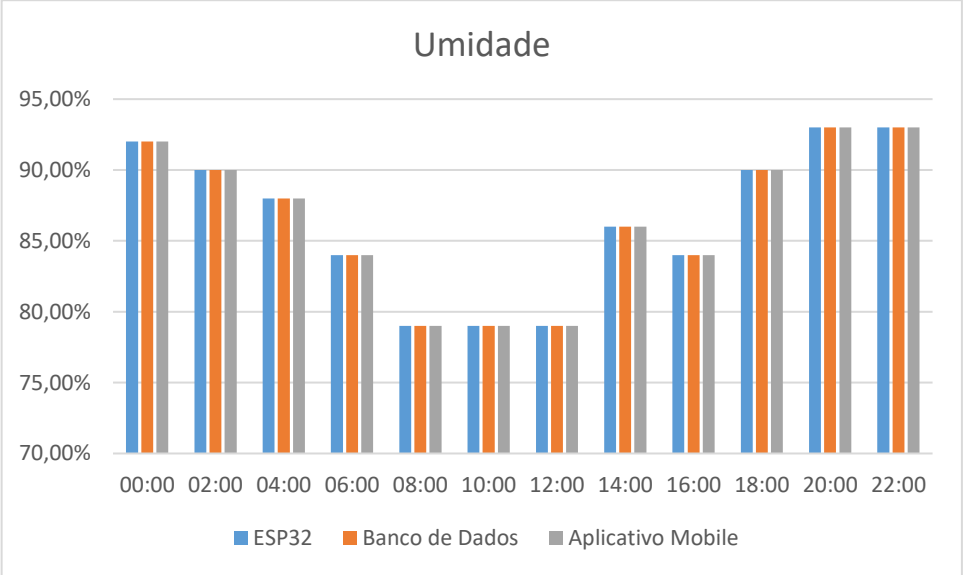
Figura 45 - Validação do fluxo de dados de temperatura



Fonte: Autoria própria (2022)

Como demonstrado na Figura 46, as medidas de umidade do ambiente tiveram uma diminuição considerável em uma parte do dia, contudo, este é um comportamento normal, considerando ainda que a diminuição ocorra na presença do sol.

Figura 46 - Validação do fluxo de dados de umidade

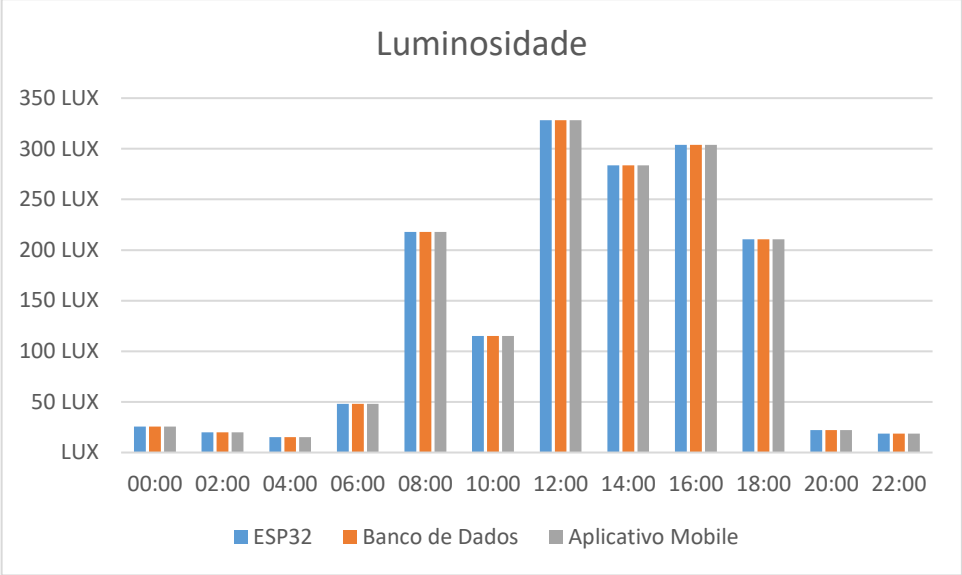


Fonte: Autoria própria (2022)

Os valores de luminosidade tiveram valores divergentes como mostrado na Figura 47, isso se dá pela grande mudança de luz no ambiente. Essas mudanças começaram a ocorrer na quinta coleta, que se dá em torno das oito horas da manhã, onde há claridade do sol justificando

esse aumento de luminosidade. A grande diferença entre a coleta do dado dez e onze se dá devido à ausência de emissores de luz, devido ao horário que foi coletado, em torno de sete horas da noite.

Figura 47 - Validação do fluxo de dados de luminosidade



Fonte: Autoria própria (2022)

5 CONCLUSÃO

O projeto constituiu no desenvolvimento de uma arquitetura para realizar o sensoriamento de aviários, através de uma leitura em tempo real de temperatura, umidade e luminosidade do ambiente e exibidas em um aplicativo mobile com o intuito de permitir que o usuário tenha acesso aos dados em qualquer lugar que tenha acesso a internet.

Os principais objetivos específicos do projeto foram alcançados, pode-se dividi-los em três principais temas: coleta de dados de sensores utilizando um microcontrolador ESP32, armazenamento desses dados em um banco de dados *Firebase* e desenvolvimento de um aplicativo *mobile Android* para exibir os gráficos em tempo real.

Com base nos resultados obtidos na seção 4.1, pode-se observar que a leitura e processamento dos sinais dos sensores ocorreu corretamente, obtendo os dados em tempo real e nos períodos programados.

Na seção 4.2, pode-se observar que a comunicação com o banco de dados ocorreu com êxito, armazenando corretamente os dados obtidos através dos sensores.

De forma similar, as seções 4.3 e 4.4 demonstram o êxito do desenvolvimento do aplicativo *mobile* e a exibição dos dados que são lidos do banco de dados, tanto em tempo real quanto na criação de relatórios automáticos dos dados obtidos no dia anterior.

Os principais desafios do projeto foram as comunicações entre o ESP32 e o banco de dados para envio de dados corretamente e a comunicação entre o banco de dados e o Aplicativo *Mobile* para leitura. Por conta das rápidas atualizações de versões, existem muitas bibliotecas e muitas formas de realizar essa comunicação, o que acaba tornando métodos de um ano atrás, por exemplo, obsoletos. Outro grande desafio foi a criação de relatórios do sistema, visto que inicialmente o projeto era composto apenas em leitura de dados em tempo real, perdendo os dados instantaneamente pelo novo valor do sensor, o qual foi necessária a criação de mais um banco de dados (*Firebase* - *Firestore*) para armazenar os valores.

Dado o êxito do funcionamento de todas as partes do sistema, pode-se concluir que o projeto funcionou corretamente, coletando, transferindo, armazenando e exibindo os dados para o usuário, tanto em tempo real quanto no relatório com o histórico destes dados.

Para melhorias na utilização do protótipo proposto, algumas adaptações podem ser feitas como a inclusão de uma bateria para alimentação externa possibilitando a movimentação do local de sensoriamento, inclusão de novos sensores, como sensor de gás, ruídos, amônia, entre outros para assegurar diferentes ambientes. Também é possível inserir um sistema de controle, ligando dispositivos como ar-condicionado, ventiladores e lâmpadas com a criação de

botões no aplicativo que ligam/desligam esses dispositivos manualmente e de forma automática conforme níveis pré-definidos pelo usuário, criação de alertas dos níveis atuais dos sensores entre outras possíveis melhorias.

REFERÊNCIAS

ABPA - Associação brasileira de proteína animal. **Relatório anual 2020**. Disponível em: https://abpa-br.org/wp-content/uploads/2020/05/abpa_relatorio_anual_2020_portugues_web.pdf. Acesso em: 16 maio 2022.

AMARAL, A. G. *et al*. Efeito do ambiente de produção sobre frangos de corte sexados criados em galpão comercial. **Arquivo brasileiro de medicina veterinária e zootecnia**, [s. l.], ano 2011, v. 63, n. 3, p. 649-658, 1 jun. 2011. DOI https://doi.org/10.1590/S0102-09352011000300017. Disponível em: https://www.scielo.br/j/abmvz/a/7cwR7C9TDSfhPC68ybbQHny/?lang=pt#. Acesso em: 23 jun. 2022.

ATZORI, L. *et al*. The *internet of things*: A survey. **Computer networks**, Calabria, p. 1-19, 31 maio 2010. DOI 10.1016/ j.comnet.2010.05.010. Disponível em: https://www.cs.mun.ca/courses/cs6910/IoT-Survey-Atzori-2010.pdf. Acesso em: 16 maio 2022.

BELUSSO, D; HESPANHOL, A. A evolução da avicultura industrial brasileira e seus efeitos territoriais. **Percurso**, Maringá, PR, v. 2, n. 1, p. 25-51, 9 jul. 2010. Disponível em: https://periodicos.uem.br/ojs/index.php/Percurso/article/view/49458. Acesso em: 16 maio 2022.

BELLAVER, C. *et al*. Boas práticas de produção de produção de frangos. **Circular técnica**, Concórdia, SC, p. 1-12, 1 out. 2003. Disponível em: https://www.infoteca.cnptia.embrapa.br/infoteca/bitstream/doc/441957/1/CUsersPiazzonDocumentsCIT38.pdf. Acesso em: 16 maio 2022.

BRITO, F. **Sensores e atuadores: controle e processos industriais**. São Paulo: Érica, 2017.

CARRION, P.; QUARESMA, M. Internet da Coisas (IoT): Definições e aplicabilidade aos usuários finais. **Internet da coisas (IoT)**: Definições e aplicabilidade aos usuários finais, P&D Designe 2018, v. 8, ed. 15, 2019. DOI https://doi.org/10.5965/2316796308152019049. Disponível em: https://www.revistas.udesc.br/index.php/hfd/article/view/2316796308152019049. Acesso em: 21 ago. 2021.

CASTRO, S. **Sensores de umidade: caracterização e desenvolvimento de dispositivo eletrônico**. 2011. Dissertação (Mestrado em Materiais para Engenharia) – Instituto de Ciências Exatas, Universidade Federal de Itajubá, Minas Gerais, 2011. Disponível em: https://repositorio.unifei.edu.br/jspui/handle/123456789/1394. Acesso em: 6 maio 2022.

GUBBI, J.; BUYYA, R.; MARUSIC, S.; PALANISWAMI, M. ***Internet of things*** **(IoT):** A vision, architectural elements, and future directions. Elsevier B.V: Future Generation Computer Systems, 2013, p.1645-1660. Disponível em < https://www.sciencedirect.com/science/article/pii/S0167739X13000241>. Acesso em: 03 fev. 2022.

KURNIAWAN, A. ***Internet of things projects with* ESP32**. [S. l.]: Packt, 2019. 232 p.

MENEGALI, I. **Diagnóstico da qualidade do ar na produção de frangos de corte em instalações semi-climatizadas por pressão negativa e positiva, no inverno, no sul do Brasil**. 2005. 96 f. Tese (Pós-Graduação em Engenharia Agrícola) - Curso de Engenharia Agrícola, Universidade Federal de Viçosa, Viçosa, 2005. Disponível em: http://arquivo.ufv.br/dea/ambiagro/arquivos/Tese%20de%20Irene%20Menegali2005.pdf. Acesso em: 24 out. 2020.

MOON, B. ***Internet of things & hardware industry overview* 2016**. SparkLabs Global Ventures, 2016.

MOUSER ELECTRONICS. ***Digital-output relative humidity & temperature sensor/module DHT11***. 2011. Disponível em: https://br.mouser.com/datasheet/2/830/DFR0067_DS_10_en-2488783.pdf. Acesso em: 16 maio 2022.

OLIVEIRA, M. E. **Implementação e avaliação de um sistema automatizado de monitoramento e controle térmico em um aviário convencional utilizando tecnologia IoT**. 2019. 140 f. Tese (Doutorado em Ciências de Engenharia Ambiental) - Curso de Engenharia Ambiental, Universidade de São Paulo, São Carlos, 2019. Disponível em: https://teses.usp.br/teses/disponiveis/18/18139/tde-30092019-094433/pt-br.php. Acesso em: 24 out. 2020.

OLIVEIRA, R. D., Donzele, J. L., Abreu, M. D., Ferreira, R. A., Vaz, R. G. M. V., & Cella, P. S. 2006. **Efeitos da temperatura e da umidade relativa sobre o desempenho e o rendimento de cortes nobres de frangos de corte de 1 a 49 dias de idade**. Revista Brasileira de Zootecnia, 35(3), 797-803.

RS. ***Light dependente resistor Datasheet.*** [S.l], 1997. Disponível em: < https://components101.com/sites/default/files/component_datasheet/LDR%20Datasheet.pdf>. Acesso em: 03/02/2022.

SANTOS, B. P. *et al*. **Internet das coisas: da teoria à prática**. In: SBRC-Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos. 2016, Santa Catarina: Anais UFMG, 2016. p. 2-51.

SANTOS, D. C. *et al*. **Prototipo para o controle da temperatura de um aviário utilizando Microcontrolador de Baixo Custo**. Computer On The Beach 2014: Resumos Expandidos. Mato Grosso do Sul, p. 418-419. jul. 2014.

SANTOS, J. W.; JUNIOR, R. C. L. **Sistema de automatização residencial de baixo custo controlado pelo microcontrolador ESP32 e monitorado via smartphone.** 2019. 46f. Trabalho de Conclusão de Curso (Técnico em Automação Industrial) – Departamento de Eletrônica, Universidade Tecnológica Federal do Paraná. Disponível em: http://repositorio.utfpr.edu.br/jspui/bitstream/1/16960/1/PG_COAUT_2019_1_02.pdf. Acesso em: 14 mar. 2021.

SCHILDT, H. **C completo e total**. 3. ed. São Paulo: McGraw-Hill, 1997.

SWAMY, S. N.; JADHAV, D.; KULKARNI, N. Security threats in the application layer in IOT applications. ***International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)***, fev. 2017.

RAMEZ, R. *et al*. **Sistemas de banco de dados.** 4. ed. rev. [S. l.]: Pearson, 2005.

THOMAZINI, D.; ALBUQUERQUE, P. U. B. **Sensores industriais: fundamentos e aplicações**. 4. ed. São Paulo: Erica, 2009.

TINÔCO, I.F.F. **Avicultura industrial: novos conceitos de materiais, concepções e técnicas construtivas disponíveis para galpões avícolas brasileiros**. Revista Brasileira de Ciência Avícola, Campinas, v.3, n.1, p.1–26, 2001.

WELKER, J. S.; ROSA, A. P.; MOURA, D. J.; MACHADO, L. P.; CATELAN, F.; UTTPATEL, R. Temperatura corporal de frangos de corte em diferentes sistemas de climatização. **Revista brasileira de zootecnia**, [S.L.], v. 37, n. 8, p. 1463-1467, ago. 2008. FapUNIFESP (SciELO). http://dx.doi.org/10.1590/s1516-35982008000800018.

WENDLING, M. Sensores. **Unesp**, [S. l.], v. 2, p. 1-19, 1 jun. 2010. Disponível em: https://www.feg.unesp.br/Home/PaginasPessoais/ProfMarceloWendling/4---sensores-v2.0.pdf. Acesso em: 16 maio 2022.

APÊNDICE A – Códigos de programação ESP32

Código Sensoriamento – IDE Arduino

```
#include <Arduino.h>
#include "DHT.h"
#define DHTPIN 27 // Define que OUT será conectado em D27
#define DHTTYPE DHT11 // Define o modelo de sensor como DHT 11
#include <Adafruit_Sensor.h>
#if defined(ESP32)
#include <WiFi.h>
#elif defined(ESP8266)
#include <ESP8266WiFi.h>
#endif
#include <Firebase_ESP_Client.h>
#define ldr 33
#include "addons/TokenHelper.h"
#include "addons/RTDBHelper.h"
//#define WIFI_SSID "VIVOFIBRA-A6B0"
//#define WIFI_PASSWORD "772D142D8D"
#define WIFI_SSID "Rodolfo 304"
#define WIFI_PASSWORD "pedeprorodolfo"
#define API_KEY "AIzaSyCv1i57Z47VNPvqsvahx7eJOu7ZHq0DNTw"
#define DATABASE_URL "monitoramentoav-bd55e-default-rtdb.firebaseio.com"
#define FIREBASE_PROJECT_ID "monitoramentoav-bd55e"
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;
DHT dht(DHTPIN, DHTTYPE);
unsigned long sendDataPrevMillis = 0;
unsigned long sendDataPrevMillis2 = 0;
int intValue;
float floatValue;
bool signupOK = false;
int count = 0;
void setup() {
Serial.begin(115200);
dht.begin();
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
Serial.print("Connecting to Wi-Fi");
while (WiFi.status() != WL_CONNECTED) {
Serial.print(".");
delay(300);
}
Serial.println();
Serial.print("Connected with IP: ");
Serial.println(WiFi.localIP());
Serial.println();
config.api_key = API_KEY;
config.database_url = DATABASE_URL;
if (Firebase.signUp(&config, &auth, "", "")) {
Serial.println("ok");
```

```
signupOK = true;
}
else {
Serial.printf("%s\n", config.signer.signupError.message.c_str());
}
config.token_status_callback = tokenStatusCallback;
Firebase.begin(&config, &auth);
Firebase.reconnectWiFi(true);
pinMode(ldr, INPUT);
}
void loop() {
float humidity = dht.readHumidity();
float temperature = dht.readTemperature();
float luminosity = analogRead(ldr);
if (Firebase.ready() && signupOK && (millis() - sendDataPrevMillis > 3000 ||
sendDataPrevMillis == 0)) {
sendDataPrevMillis = millis();
Serial.print("Luminosidade: ");
Serial.println(luminosity);
Serial.print("Temperatura: ");
Serial.println(temperature);
Serial.print("Umidade: ");
Serial.println(humidity);
Serial.println("\n\n");
if(millis() - sendDataPrevMillis2 > 7200000 || sendDataPrevMillis2 == 0) {
sendDataPrevMillis2 = millis();
if (count < 22 ){
    a. count = count + 2;
}else {
    a. count = 0;
}
FirebaseJson content;
String documentPath = "sensorData/sensor"+String(count);
content.set("fields/Temperature/doubleValue", String(temperature).c_str());
content.set("fields/Humidity/doubleValue", String(humidity).c_str());
content.set("fields/Luminosity/doubleValue", String(luminosity).c_str());
if(Firebase.Firestore.patchDocument(&fbdo, FIREBASE_PROJECT_ID, "",
documentPath.c_str(), content.raw(), "Temperature,Humidity,Luminosity")){
    a. Serial.printf("ok\n%s\n\n", fbdo.payload().c_str());
    b. return;
}else{
    a. Serial.println(fbdo.errorReason());
}
}
Firebase.RTDB.setFloat(&fbdo, "/User/Luminosidade", luminosity);
if (Firebase.RTDB.getFloat(&fbdo, "/User/Luminosidade")) {
if (fbdo.dataType() == "float") {
floatValue = fbdo.floatData();
}
}
```

```
92. else {
93. Serial.println(fbdo.errorReason());
94. }
95. Firebase.RTDB.setFloat(&fbdo, "/User/Temperatura", temperature);
96. if (Firebase.RTDB.getFloat(&fbdo, "/User/Temperatura")) {
97. if (fbdo.dataType() == "float") {
98. floatValue = fbdo.floatData();
99. }
100.        }
101.        else {
102.        Serial.println(fbdo.errorReason());
103.        }
104.        Firebase.RTDB.setFloat(&fbdo, "/User/Umidade", humidity);
105.        if (Firebase.RTDB.getFloat(&fbdo, "/User/Umidade")) {
106.        if (fbdo.dataType() == "float") {
107.        floatValue = fbdo.floatData();
108.        }
109.        }
110.        else {
111.        Serial.println(fbdo.errorReason());
112.        }
113.        }
114.        }
```

APÊNDICE B – Código de programação Aplicativo Mobile

Código Root - App.js

```
import React, { useState, useEffect } from 'react';
import { Button, TextInput, Image, Text, View } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import Home from './Pages/Home';
import Login from './Pages/Login';
import Relatorio from './Pages/Relatorio/relatorio';
import Configuracao from './Pages/Configuracao/config';
const Stack = createNativeStackNavigator()
function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Login">
        <Stack.Screen
          name="Login" component={Login}
          options={{
            headerShown: false
          }} />
        <Stack.Screen
          name="Home" component={Home}
          options={{
            headerShown: false
          }} />
        <Stack.Screen name="Relatorio" component={Relatorio} />
        <Stack.Screen name="Configuracao" component={Configuracao} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
export default App;
```

Código Root - index.js

```
@format
import {AppRegistry} from 'react-native';
import App from './App';
import {name as appName} from './app.json';
AppRegistry.registerComponent(appName, () => App);
```

Código Firebase - firebase.js

```
import { initializeApp } from 'firebase/app';
import firebase from 'firebase';
import 'firebase/firestore';
import 'firebase/auth';
import 'firebase/storage';
import database from "@react-native-firebase/database";
import firestore from "@react-native-firebase/firestore";
```

```
const firebaseConfig = {
    apiKey: 'AIzaSyCv1i57Z47VNPvqsvahx7eJOu7ZHq0DNTw',
    authDomain: 'monitoramentoav-bd55e.firebaseapp.com',
    databaseURL: 'https://monitoramentoav-bd55e-default-rtdb.firebaseio.com',
    projectId: 'monitoramentoav-bd55e',
    storageBucket: 'monitoramentoav-bd55e.appspot.com',
    messagingSenderId: '854034421704',
    appId: '1:854034421704:web:a09eaf487afb85c25252fd',
};
firebase.initializeApp(firebaseConfig);
const db = firebase.firestore();
const auth = firebase.auth();
const sensorData = db.collection('sensorData');
export { firebase, firestore, database, auth, sensorData };
```

Código Login – index.js

```
import React, { useState, useEffect } from 'react';
import 'react-native-gesture-handler';
import { Text, TextInput, Image, TouchableOpacity, StyleSheet,
KeyboardAvoidingView, View } from 'react-native';
import { firebase, auth } from '../../services/firebase';
function Login({ navigation }) {
    const loginFirebase = () => {
        firebase.auth().signInWithEmailAndPassword(email, password)
            .then((userCredential) => {
                // Signed in
                let user = userCredential.user;
                navigation.navigate('Home')
            })
            .catch((error) => {
                setErrorLogin("true")
                let errorCode = error.code;
                let errorMessage = error.message;
            })
    }
    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const [errorLogin, setErrorLogin] = useState("");
    return (
        <KeyboardAvoidingView style={styles.background}>
            <View>
                <Image source={require('../../assets/frangologo1.png')} style={styles.logo}
/>
            </View>
            <View>
                <Text style={styles.textoMonitoramento}> MonitoramentoAV </Text>
            </View>
            <KeyboardAvoidingView style={styles.botoes}>
                <TextInput
```

```
32.                 placeholder=" EMAIL "
33.                 autoCorrect={false}
34.                 style={styles.inputBotao}
35.                 onChangeText={(text) => setEmail(text)}
36.                 value={email}
37.             />
38.             <TextInput
39.                 placeholder="SENHA"
40.                 autoCorrect={false}
41.                 style={styles.inputBotao}
42.                 secureTextEntry={true}
43.                 onChangeText={(text) => setPassword(text)}
44.                 value={password}
45.             />
46.             <KeyboardAvoidingView>
47.                 <TouchableOpacity
48.                     onPress={loginFirebase}
49.                     style={styles.buttonLogin}>
50.                     <Text style={styles.textoButtonLogin}> Login </Text>
51.                 </TouchableOpacity>
52.             </KeyboardAvoidingView>
53.         </KeyboardAvoidingView>
54.     </KeyboardAvoidingView>
55.   );
56. }
57. const styles = StyleSheet.create({
58.   background: {
59.     backgroundColor: 'black',
60.     flex: 1,
61.   },
62.   logo: {
63.     alignContent: 'center',
64.     alignSelf: 'center',
65.     width: '50%',
66.     height: '50%',
67.     marginTop: 100,
68.   },
69.   textoMonitoramento: {
70.     flex: 1,
71.     marginTop: -150,
72.
73.     fontSize: 17,
74.     color: "#fff",
75.     fontWeight: "bold",
76.     alignSelf: "center",
77.   },
78.   botoes: {
79.     flex: 1,
80.     marginTop: -250,
81.     alignItems: 'center',
```

```
82.       justifyContent: 'center',
83.       margin: 15,
84.       padding: 10,
85.    },
86.    inputBotao: {
87.       backgroundColor: '#FFF',
88.       width: '90%',
89.       marginBottom: 15,
90.       color: '#222',
91.       fontSize: 17,
92.       borderRadius: 10,
93.       padding: 10,
94.    },
95.    buttonLogin: {
96.       backgroundColor: "#009688",
97.       borderRadius: 10,
98.       paddingVertical: 10,
99.       paddingHorizontal: 126
100.          },
101.          textoButtonLogin: {
102.             fontSize: 17,
103.             color: "#fff",
104.             fontWeight: "bold",
105.             alignSelf: "center",
106.             textTransform: "uppercase"
107.          }
108.       })
109.       export default Login;
```

Código Login – style.js

```
1.  import { StyleSheet } from 'react-native';
2.  const styles = StyleSheet.create({
3.     background: {
4.        backgroundColor: '#000000',
5.        flex: 1,
6.        justifyContent: 'center',
7.        alignItems: 'center'
8.     },
9.     logo: {
10.       width: 400,
11.       height: 400,
12.    },
13.    logoImg: {
14.       width: '50%',
15.       height: '50%'
16.    },
17.    container: {
18.       display: 'flex',
19.       flexDirection: 'column',
```

```
        color: '#FFFFFF'
    },
    button: {
        backgroundColor: '#1A91F0',
        color: 'black',
        width: 400,
        height: '200px',
    },
});
export default styles;
```

Código Home – index.js

```
import React from 'react';
import Sensor from './sensor';
import { TouchableOpacity, TextInput, Image, Text, View, StyleSheet } from 'react-native';
import 'react-native-gesture-handler';
function Home({ navigation }) {
    return (
        <View style={styles.background}>
            <View style={styles.botoesPosicao}>
                <TouchableOpacity onPress={() => navigation.navigate('Relatorio')} style={styles.button}>
                    <Text style={styles.textButton}> Realtório dia anterior </Text>
                </TouchableOpacity>
                <TouchableOpacity onPress={() => navigation.navigate('Login')} style={styles.button}>
                    <Text style={styles.textButton}> Sair </Text>
                </TouchableOpacity>
            </View>
            <Sensor />
        </View>
    );
}
const styles = StyleSheet.create({
    background: {
        flex: 1,
    },
    button: {
        marginBottom: 15,
        color: '#222',
        padding: 10,
    },
    botoesPosicao: {
        alignItems: "baseline",
        flexDirection: "row",
        justifyContent: "center",
        margin: 0,
        padding: 10,
```

```
35.    },
36.    textButton: {
37.        backgroundColor: "#FFF",
38.        width: 180,
39.        height: 30,
40.        borderWidth: 2,
41.        fontSize: 13,
42.        color: "#000",
43.        alignSelf: "center",
44.        justifySelf: "center",
45.        textTransform: "uppercase",
46.        textAlign: "center",
47.        padding: 5
48.    }
49. })
50. export default Home;
```

Código Home – sensor.js

```
1.  import React, { useEffect, useState } from 'react';
2.  import { View, Text, StyleSheet } from 'react-native';
3.  import { database, firestore, sensorData } from '../../services/firebase';
4.  export const ArrayTPT = new Array;
5.  export const ArrayHUMI = new Array;
6.  export const ArrayLUMI = new Array;
7.  export let valueTPT = 0;
8.  export let valueHUMI = 0;
9.  export let valueLUMI = 0;
10. function Sensor() {
11.    const [sensorValues, setSensorValues] = useState({})
12.    useEffect(() => {
13.        database()
14.            .ref('User')
15.            .on('value', values => {
16.                setSensorValues(values.val())
17.            });
18.    }, [])
19.    const DataArray = [];
20.    const [SensorData, setSensorData] = useState({})
21.    useEffect(() => {
22.        firestore()
23.            .collection('sensorData')
24.            .get()
25.            .then(querySnapshot => {
26.                querySnapshot.forEach(documentSnapshot => {
27.                    setSensorData(documentSnapshot.data())
28.                    console.log(documentSnapshot.data());
29.                    DataArray.push({ "Temperature": documentSnapshot.data().Temperature,
    "Luminosity": documentSnapshot.data().Luminosity, "Humidity":
    documentSnapshot.data().Humidity })
```

```
30.                 console.log("isso é o data insert", DataArray)
31.                 ArrayTPT.push(documentSnapshot.data().Temperature)
32.                 ArrayLUMI.push(documentSnapshot.data().Luminosity)
33.                 ArrayHUMI.push(documentSnapshot.data().Humidity)
34.                 console.log("Array da TPT: ", ArrayTPT)
35.                 console.log("Array da HUMI: ", ArrayHUMI)
36.                 console.log("Array da LUMI: ", ArrayLUMI)
37.             })
38.         })
39.     }, [])
40.     valueTPT = sensorValues.Temperatura;
41.     valueHUMI = sensorValues.Humidity;
42.     valueLUMI = sensorValues.Luminosity;
43.     return (
44.         <View style={styles.monitor}>
45.             <Text style={styles.textoNomeSensor}>Temperatura:
46.             </Text>
47.             <Text style={styles.textoSensor}>
48.                 {sensorValues.Temperatura}
49.             </Text>
50.             <Text style={styles.textoNomeSensor}>Umidade:
51.             </Text>
52.             <Text style={styles.textoSensor}>
53.                 {sensorValues.Umidade}
54.             </Text>
55.             <Text style={styles.textoNomeSensor}>Luminosidade:
56.             </Text>
57.             <Text style={styles.textoSensor}>
58.                 {sensorValues.Luminosidade}
59.             </Text>
60.         </View>
61.     )
62. }
63. const styles = StyleSheet.create({
64.     monitor: {
65.         margin: 15,
66.         borderRadius: 5,
67.         padding: 10,
68.         alignItems: "flex-start",
69.         justifyContent: "center",
70.     },
71.     textoSensor: {
72.         marginLeft: 10,
73.         fontSize: 26,
74.         padding: 5,
75.         fontWeight: 'bold'
76.     },
77.     textoNomeSensor: {
78.         fontSize: 24,
79.         fontWeight: 'bold',
```

```
80.         padding: 5,
81.     }
82. })
83. export default Sensor;
```

Código Relatórios – relatorio.js

```
1.  import React from 'react';
2.  import { View, StyleSheet, Text, ScrollView } from 'react-native';
3.  import 'react-native-gesture-handler';
4.  import { LineChart, YAxis, XAxis, Grid } from 'react-native-svg-charts'
5.  import { ArrayHUMI, ArrayTPT, ArrayLUMI } from '../Home/sensor.js';
6.  function Relatorio({ navigation }) {
7.      const dataTPT = [ArrayTPT[0], ArrayTPT[6], ArrayTPT[9], ArrayTPT[10],
    ArrayTPT[11], ArrayTPT[1], ArrayTPT[2], ArrayTPT[3], ArrayTPT[4],
    ArrayTPT[5], ArrayTPT[7], ArrayTPT[8]]
8.      const dataHUMI = [ArrayHUMI[0], ArrayHUMI[6], ArrayHUMI[9],
    ArrayHUMI[10], ArrayHUMI[11], ArrayHUMI[1], ArrayHUMI[2], ArrayHUMI[3],
    ArrayHUMI[4], ArrayHUMI[5], ArrayHUMI[7], ArrayHUMI[8]]
9.      const dataLUMI = [ArrayLUMI[0], ArrayLUMI[6], ArrayLUMI[9],
    ArrayLUMI[10], ArrayLUMI[11], ArrayLUMI[1], ArrayLUMI[2], ArrayLUMI[3],
    ArrayLUMI[4], ArrayLUMI[5], ArrayLUMI[7], ArrayLUMI[8]]
10.     const contentInset = { top: 20, bottom: 20 }
11.     const axesSvg = { fontSize: 10, fill: 'grey' };
12.     const verticalContentInset = { top: 10, bottom: 10 }
13.     return (
14.         <ScrollView style={styles.background}>
15.             <Text style={{ textAlign: "center", fontWeight: "bold", fontSize: 15
    }}>Temperatura</Text>
16.             <View style={{ height: 200, padding: 5, flexDirection: 'row' }}>
17.                 <YAxis
18.                     style={{ paddingBottom: 30 }}
19.                     data={dataTPT}
20.                     contentInset={contentInset}
21.                     svg={{
22.                         fill: 'grey',
23.                         fontSize: 10,
24.                     }}
25.                     numberOfTicks={10}
26.                     formatLabel={(value) => `${value}ºC`}
27.                 />
28.                 <View style={{ flex: 1, marginLeft: 10 }}>
29.                     <LineChart
30.                         style={{ flex: 1, marginLeft: 16 }}
31.                         data={dataTPT}
32.                         svg={{ stroke: 'rgb(134, 65, 244)' }}
33.                         contentInset={contentInset}
34.                     >
35.                         <Grid />
36.                     </LineChart>
```

```
37.                 <XAxis
38.                     style={{ marginHorizontal: 5, height: 30 }}
39.                     data={dataTPT}
40.                     formatLabel={(value, index) => index}
41.                     contentInset={{ left: 10, right: 10 }}
42.                     svg={axesSvg}
43.                     formatLabel={(value) => `${value * 2}`}
44.                 />
45.             </View>
46.         </View>
47.         <Text style={{ textAlign: "center", fontWeight: "bold", fontSize: 15
    }}>Luminosidade</Text>
48.         <View style={{ height: 200, padding: 5, flexDirection: 'row' }}>
49.             <YAxis
50.                 style={{ paddingBottom: 30 }}
51.                 data={dataLUMI}
52.                 contentInset={contentInset}
53.                 svg={{
54.                     fill: 'grey',
55.                     fontSize: 10,
56.                 }}
57.                 numberOfTicks={10}
58.                 formatLabel={(value) => `${value} lux`}
59.             />
60.             <View style={{ flex: 1, marginLeft: 10 }}>
61.                 <LineChart
62.                     style={{ flex: 1, marginLeft: 16 }}
63.                     data={dataLUMI}
64.                     svg={{ stroke: 'rgb(134, 65, 244)' }}
65.                     contentInset={contentInset}
66.                 >
67.                     <Grid />
68.                 </LineChart>
69.                 <XAxis
70.                     style={{ marginHorizontal: 5, height: 30 }}
71.                     data={dataLUMI}
72.                     formatLabel={(value, index) => index}
73.                     contentInset={{ left: 10, right: 10 }}
74.                     svg={axesSvg}
75.                     formatLabel={(value) => `${value * 2}`}
76.                 />
77.             </View>
78.         </View>
79.         <Text style={{ textAlign: "center", fontWeight: "bold", fontSize: 15
    }}>Umidade</Text>
80.         <View style={{ height: 200, padding: 5, flexDirection: 'row' }}>
81.             <YAxis
82.                 style={{ paddingBottom: 30 }}
83.                 data={dataHUMI}
84.                 contentInset={contentInset}
```

```
85.                 svg={{
86.                     fill: 'grey',
87.                     fontSize: 10,
88.                 }}
89.                 numberOfTicks={10}
90.                 formatLabel={(value) => `${value}%`}
91.             />
92.             <View style={{ flex: 1, marginLeft: 10 }}>
93.                 <LineChart
94.                     style={{ flex: 1, marginLeft: 16 }}
95.                     data={dataHUMI}
96.                     svg={{ stroke: 'rgb(134, 65, 244)' }}
97.                     contentInset={contentInset}
98.                 >
99.                     <Grid />
100.                     </LineChart>
101.                     <XAxis
102.                         style={{ marginHorizontal: 5, height: 30 }}
103.                         data={dataHUMI}
104.                         formatLabel={(value, index) => index}
105.                         contentInset={{ left: 10, right: 10 }}
106.                         svg={axesSvg}
107.                         formatLabel={(value) => `${value * 2}`}
108.                     />
109.                 </View>
110.             </View>
111.         </ScrollView>
112.     );
113. }
114. const styles = StyleSheet.create({
115.     background: {
116.         flex: 1,
117.         margin: 7,
118.     },
119.     graphTemp: {
120.         backgroundColor: '#FFF',
121.         marginBottom: 15,
122.         height: 200,
123.         flexDirection: 'row',
124.     },
125.     graphLumi: {
126.         backgroundColor: '#FFF',
127.         marginBottom: 15,
128.         height: 200,
129.         flexDirection: 'row',
130.     },
131.     graphHumi: {
132.         backgroundColor: '#FFF',
133.         marginBottom: 15,
134.         height: 200,
```

```
135.                flexDirection: 'row',
136.            }
137.        })
138.        export default Relatorio;
```