

Monitoramento de colmeias com Internet das Coisas

J. C. Gonçalves D. M. Dos Santos

L. J. S. Dos S. P. Monroe V. A. Scholze F. Fruett

L. F. Bittencourt

Relatório Técnico - IC-PFG-24-03

Projeto Final de Graduação

2024 - Julho

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

The contents of this report are the sole responsibility of the authors.
O conteúdo deste relatório é de única responsabilidade dos autores.

Monitoramento de colmeias com Internet das Coisas

Jonas C. Gonçalves* Daniel M. Dos Santos† Lindon J. S. Dos S. P. Monroe‡
Victor A. Scholze§ F. Fruett¶ Luiz F. Bittencourt||

Resumo

Este é o relatório do projeto realizado como Trabalho de Conclusão de Curso do Instituto de Computação, em parceria com o Prof. Roberto Greco do Instituto de Geociências e o Prof. Fabiano Fruett da Faculdade de Engenharia Elétrica e de Computação, cujo objetivo é desenvolver um sistema para coleta de dados de colmeias que serão instalados em escolas para o aprendizado dos alunos e alunas.

O sistema faz a coleta de dados do ambiente da colmeia via rede sem fio com uma placa embarcada, à distância das próprias abelhas, assim mantendo a segurança delas e a integridade da placa. O projeto teve quatro placas fornecidas, além dos dois sensores, pelo Prof. Fabiano. Os sensores realizam medição da temperatura, umidade, som, pressão e proximidade. Ademais condições climáticas são advindas da API OpenWeatherMap.

1 Introdução

Em 2022, o professor Roberto Greco, do Instituto de Geografia da UNICAMP, iniciou um projeto para a instalação de colmeias no assentamento Milton Santos, em Americana, SP. Como parte deste projeto, alguns alunos do Instituto de Computação da UNICAMP, sob a orientação do professor Luiz Fernando Bittencourt, desenvolveram uma ferramenta para monitorar as colmeias como trabalho de conclusão de curso[24]. Esta ferramenta consistia em uma placa com um microcontrolador ESP32 que lia sensores de temperatura, umidade e som, enviando os dados por Bluetooth para um aplicativo desenvolvido por eles.

Pouco após a implementação inicial, o projeto encontrou alguns problemas. Para resolver essas questões, o professor Roberto Greco buscou a ajuda do professor Fabiano Fruett, da Faculdade de Engenharia Elétrica e Computação da UNICAMP. Fruett desenvolveu uma nova versão da placa, desta vez utilizando uma Raspberry Pi Pico W, equipada com um microprocessador RP2040 com 2MB de memória flash e capacidade de conexão WiFi e

*j256453@dac.unicamp.br

†d214752@dac.unicamp.br

‡l220407@dac.unicamp.br

§v206457@dac.unicamp.br

¶fabiano@unicamp.br

||bit@ic.unicamp.br

Bluetooth. A nova placa também incorporava sensores adicionais de temperatura, umidade, pressão, proximidade e som.

Em 2023, com o objetivo de aprimorar o sistema de monitoramento de colmeias e expandir sua aplicação para escolas, o projeto foi atualizado com a colaboração de Marcos C. Rosa, Lucas C. Castello e Carlos A. A. Trujillo, sob a orientação do professor Luiz F. Bittencourt, do Instituto de Computação da UNICAMP. A nova versão do sistema foi desenvolvida utilizando uma placa equipada com sensores de temperatura, umidade, som e proximidade, fornecida pelo professor Fabiano Fruett da Faculdade de Engenharia Elétrica e Computação. Este sistema melhorado faz uso da tecnologia de Internet das Coisas (IoT) para permitir a comunicação via Wi-Fi entre a placa e um aplicativo Android, possibilitando o monitoramento remoto das colmeias instaladas em escolas.

O desenvolvimento do projeto seguiu uma metodologia estruturada, dividindo-se em duas frentes principais: a programação do microcontrolador, e a criação do aplicativo móvel. A comunicação entre o microcontrolador e o aplicativo foi implementada utilizando o protocolo HTTP/1.1, permitindo requisições GET e POST. Os dados coletados pelos sensores são transmitidos via Wi-Fi, armazenados em um cartão microSD e apresentados no aplicativo por meio de gráficos, facilitando a visualização das métricas de temperatura, umidade, som e movimentação das abelhas[26].

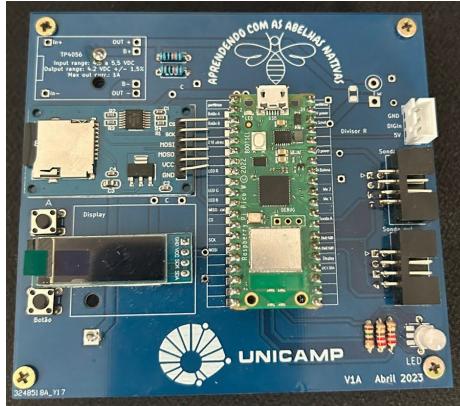


Figura 1: Placa elaborada para monitoramento de colmeias nos projetos anteriores

No entanto, apesar das melhorias, o projeto enfrentou novos desafios, a comunicação WiFi, embora em grande parte funcional, apresentou alguns erros de conexão. Além disso, apenas o sensor de proximidade funcionou satisfatoriamente, enquanto os sensores de som e o BME680 não operaram corretamente devido a problemas de implementação do software desenvolvido pelos integrantes. Houve também limitações relacionadas à capacidade de memória do microcontrolador, o que exigiu soluções alternativas, como o envio de dados em pequenos pacotes. Houve um planejamento de desenvolvimento futuro do projeto que incluiria a integração adequada dos sensores restantes e a expansão da funcionalidade do aplicativo para uma versão desktop na Web ou em um executável.

Em 2024, já com as experiências e relatos dos membros dos projetos anteriores, foi realizada uma reunião com o professor Fabiano Fruett para alinhar as dificuldades apresentadas

pelas equipes anteriores e discutir a possível requisição de novas placas para todos os membros do grupo. Durante a reunião, foi apresentada uma placa mais robusta e de código aberto, já utilizada em outros projetos com objetivos educacionais. Considerando os problemas com a antiga placa e a falta de unidades suficientes (apenas uma placa para quatro membros), decidiu-se substituir a antiga placa pela BitDogLab, desenvolvida pelo professor Fruett, além disso foram disponibilizadas quatro placas para o grupo, garantindo-se uma para cada membro. A ideia do uso dessa placa é a mesclagem dos projetos pedagógicos propostos pelas duas placas em uma única e, a bitDogLab é perfeita para esse objetivo, pois apresenta múltiplas outras features e faz parte de uma comunidade com outros contribuidores.

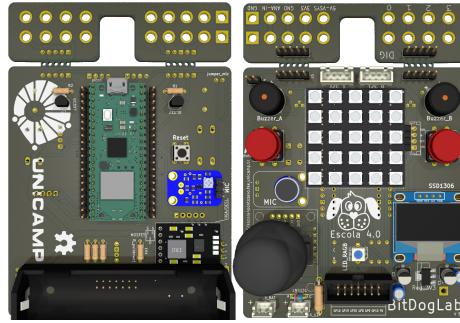


Figura 2: Placa BitDogLab na revisão v5.4

Este relatório detalha a programação da placa confeccionada para monitorar as colmeias, que desta vez será instalada em escolas. Além disso, descreve a tentativa de desenvolvimento de um novo aplicativo para comunicação com o microcontrolador, ambos baseados no conceito de Internet das Coisas (IoT) e no projeto do ano anterior.

Está estruturado nas seções abaixo que abordam de maneira detalhada os diversos aspectos do projeto. As seções são organizadas da seguinte forma:

- Objetivos: Esta seção apresenta os objetivos gerais e específicos do projeto, delineando o propósito e os resultados esperados com a implementação do sistema de monitoramento de colmeias em escolas.
- Metodologia: Nesta seção, é descrito o planejamento e a estruturação do projeto. Inclui-se aqui a abordagem metodológica adotada, os recursos utilizados, as etapas de desenvolvimento e os procedimentos de teste e validação das soluções implementadas.
- Comunicação: Aqui é explicada a forma de comunicação entre o microcontrolador e o Firebase, as conexões que foram modificadas em relação à versão anterior do projeto, e as estratégias para garantir a confiabilidade e eficiência na transmissão das informações coletadas pelos sensores.
- Placa: Esta seção fornece uma descrição detalhada da implementação das funcionalidades da placa microcontroladora. Inclui informações sobre a escolha dos compo-

tes, a programação do firmware, e a integração dos sensores para a coleta de dados ambientais das colmeias.

- Aplicativo: Descreve o aplicativo móvel que era utilizado anteriormente para a configuração da placa e leitura dos dados enviados pelo microcontrolador.
- Documentação: Nesta seção, são apresentados os links para os repositórios que contêm o código-fonte e a documentação completa do projeto, inclui-se também informações sobre a estrutura dos repositórios.
- Resultados: Descreve os resultados obtidos com a implementação do sistema, são apresentados dados coletados durante os testes, análises dos resultados, e discussões sobre a eficácia e as limitações da solução proposta.
- Modo de Uso: Fornece instruções detalhadas sobre como utilizar o sistema de monitoramento, configuração do microcontrolador, e uso do site para acessar e interpretar os dados.
- Conclusão e Trabalhos Futuros: Nesta seção, são feitas as considerações finais sobre o projeto, discute-se o impacto dos resultados, as contribuições para o campo de estudo, e as possíveis direções para trabalhos futuros.
- Referências: Apresenta as fontes e referências utilizadas pelos autores ao longo do desenvolvimento do projeto, inclui-se aqui livros, artigos científicos, e outros materiais consultados para a fundamentação teórica e técnica do trabalho, além de links para acesso de conteúdo criado pelos autores.

2 Objetivos

Este projeto visa integrar tecnologia avançada para o monitoramento ambiental de colmeias, utilizando a placa BitDogLab, que permite uma coleta e análise detalhada dos dados ambientais das colmeias. Os objetivos estão estruturados da seguinte maneira:

2.1 Objetivo Principal

- Desenvolver um sistema integrado que utilize sensores de temperatura, umidade, som e movimentação de abelhas na colmeia, permitindo a comunicação dos dados coletados via Wi-Fi para um aplicativo Android, que irá processar e apresentar essas informações em gráficos dinâmicos.

2.2 Objetivos Específicos

Para atingir o objetivo principal, subdividimos as metas em tarefas menores, que juntas compõem o escopo completo do projeto:

- Coleta de Dados: Implementar sensores na colmeia para coletar dados contínuos de temperatura, umidade, som e atividade das abelhas, garantindo uma monitorização detalhada do habitat.
- Armazenamento de Dados: Armazenar os dados coletados na nuvem do Firebase, proporcionando um método de backup seguro e acessível para análise de dados históricos.
- Conexão sem Fio: Utilizar o módulo Wi-Fi da BitDogLab para conectar o microcontrolador à rede Wi-Fi local, mas sem precisar do suporte suporte adicional do Bluetooth Low Energy para configurações iniciais ou conexões backup.
- Transmissão de Dados: Enviar os dados coletados do microcontrolador para o Firebase via Wi-Fi, assegurando a entrega eficiente da informação.
- Visualização de Dados: Desenvolver uma página Web com a capacidade de exibir os dados ambientais através de gráficos intuitivos, facilitando a interpretação dos padrões e condições das colmeias.
- Análise de Tendências: Mostrar no aplicativo as tendências dos dados coletados, incluindo valores mínimos e máximos para cada métrica, permitindo aos usuários identificar rapidamente mudanças críticas no ambiente das colmeias.

Cada um desses objetivos específicos contribui diretamente para o sucesso do objetivo principal, formando um sistema de monitoramento robusto e informativo para a conservação e estudo das colmeias.

3 Metodologia

3.1 Divisão de Tarefas

Com base na experiência dos autores e na estrutura do projeto, que se divide naturalmente entre hardware e software, os membros da equipe foram designados para focar em áreas específicas conforme suas especializações. Foram estabelecidas reuniões semanais para discussão da interface entre as duas partes principais do projeto, relato de problemas, proposição de soluções e, quando necessário, consulta com membros antigos e também os professores para auxílio.

3.2 Divisão de Tarefas

O esquema do projeto foi organizado da seguinte forma:

- Daniel e Lindon ficaram responsáveis pela parte de hardware, com foco nos seguintes pontos:
 1. Sensores de Som: Calibrar e testar a consistência dos microfones de eletreto para assegurar uma captura precisa de sons ambiente da colmeia, focalizando na

clareza e na distinção de sons internos e externos, fundamentais para análises comportamentais e ambientais.

2. Sensor de Proximidade Ultrassônico E18-D80NK - NPN: Usado para detectar a entrada e saída de abelhas, com alcance de detecção de 3 a 80 cm, verificar a eficácia e precisão na detecção.
 3. Microfone de Eletreto Módulo MAX4466: Avaliar a captura de sons ambiente da colmeia, permitindo análises acústicas das atividades internas.
 4. Sensor Ambiental BME680: Mede temperatura, umidade, pressão atmosférica e VOCs..
- Jonas e Víctor assumiram a parte de software, encarregando-se dos aspectos seguintes:
 1. Dados Climáticos: Integrar dados climáticos da API OpenWeatherMap para complementar as leituras dos sensores locais e proporcionar uma análise ambiental mais abrangente.
 2. Visualização Web/Executável: Desenvolver uma interface web ou um aplicativo executável em Flutter para a visualização interativa dos dados coletados, oferecendo uma experiência de usuário aprimorada e facilitando o acesso às informações.
 3. Banco de Dados em Tempo Real: Implementar o uso da API do Firebase, especificamente a ferramenta Realtime Database, para armazenar e gerenciar os dados coletados em tempo real, proporcionando uma base de dados eficiente e escalável para o projeto.

Essa estruturação das responsabilidades visa maximizar a eficiência do projeto ao alinhar as habilidades dos membros da equipe com as necessidades técnicas do projeto, garantindo que todos os aspectos críticos sejam abordados com expertise adequada.

3.3 Avaliação de Componentes

As questões com os sensores serão primeiramente abordadas por meio de testes de software, para verificar se os problemas podem ser resolvidos sem a necessidade de substituição de hardware. Caso se confirme que os sensores estão defeituosos, os professores responsáveis pelo fornecimento dos componentes serão consultados para a aquisição de substituições adequadas.

4 Comunicação

Após a definição das responsabilidades das equipes, o próximo passo foi estabelecer a comunicação entre o aplicativo e a placa, ponto central que orientaria o desenvolvimento de ambas. O principal objetivo desta comunicação é permitir que a placa envie os dados ao Firebase e que o site seja capaz de ler esses dados. Além disso, a placa faz requisições à API OpenWeatherMap utilizando o método GET da biblioteca urequests.

No projeto anterior, o envio dos dados era realizado via Wi-Fi, uma vez que o destino das placas eram escolas que possuíam essa infraestrutura. Para isso, foi utilizado o protocolo HTTP/1.1, com as requisições principais sendo GET e POST, e respostas retornadas através de códigos de status(*status code*)[15].

Atualmente, o funcionamento envolve a conexão ao Firebase, o envio de dados via métodos HTTP como PUT, PATCH, GET e DELETE, e a desconexão após a conclusão das operações.L.

Além disso, para a conexão inicial da placa à rede local, que antes foi implementado um mecanismo de comunicação via Bluetooth foi retirado para evitar a dependência da placa e do aparelho com o aplicativo no mesmo lugar e também liberar o uso de recursos.

5 Placa

A implementação do hardware para o monitoramento de colmeias no seu projeto foi atualizada para usar a placa BitDogLab, desenvolvida no âmbito do Projeto Escola 4.0 da UNICAMP. A seguir, estão listados os componentes da BitDogLab e um resumo de suas funções:

5.1 Sensores e Componentes

- Raspberry Pi Pico W: Equipada com um microcontrolador RP2040, possui 2MB de memória flash, suporta conexões Wi-Fi e Bluetooth, e possui entrada micro USB B com 40 pinos para diferentes funcionalidades.
- Sensor de Proximidade Ultrassônico E18-D80NK - NPN: Usado para detectar a entrada e saída de abelhas, com alcance de detecção de 3 a 80 cm.
- Microfone de Eletreto Módulo MAX4466: Captura sons ambientais da colmeia, conectado ao GPIO28, permitindo análises acústicas das atividades internas.
- Sensor Ambiental BME680: Mede temperatura, umidade, pressão atmosférica e VOCs, fornecendo dados vitais para o estudo do ambiente da colmeia.
- LED RGB cátodo comum: Conectado aos GPIOs 12, 13, e 11 para indicação visual através de cores variadas.
- Botões A e B: Conectados aos GPIOs 5 e 6, com resistores de pull-up internos para ações de interação manual.
- Joystick analógico KY023: Fornece controle manual através dos GPIOs 26 e 27 para os eixos e um botão no GPIO 22.
- Display OLED (0.96 polegadas): Utiliza comunicação I2C, conectado aos GPIOs 14 e 15, para apresentar dados e diagnósticos.
- Buzzers A e B: Emitindo sinais sonoros através dos GPIOs 21 e 10, para alertas ou feedback operacional.

- Matriz de LEDs WS2812B: Conectada ao GPIO7, usada para efeitos visuais e indicadores de status.

5.2 Ciclo da Placa

A implementação da placa BitDogLab em um projeto de monitoramento de colmeias envolve um ciclo operacional complexo e planejado para garantir a coleta, armazenamento e análise eficaz dos dados ambientais. Este ciclo é estruturado em várias etapas, descritas a seguir:

1. Inicialização do Sistema: No momento do boot, a placa executa uma série de testes de diagnóstico para verificar a integridade dos sensores e das conexões de rede. Isso assegura que todos os componentes estejam operacionais antes de iniciar a coleta de dados, minimizando o risco de erros e dados corrompidos.
2. Leitura dos Sensores: A placa realiza leituras periódicas dos sensores ambientais incorporados, incluindo sensores de temperatura, umidade, som, gases VOC (de compostos orgânicos voláteis) e movimento, além da requisição dos dados climáticos via API. Estas leituras são programadas através de um cronograma configurável, permitindo ajustes conforme as necessidades específicas do estudo. A frequência das leituras pode ser aumentada em períodos de maior atividade das abelhas, como durante a primavera.
3. Armazenamento de Dados via Firebase: Os dados validados são então enviados para o Firebase utilizando a API da ferramenta Realtime Database. O uso dessa plataforma baseada em nuvem não apenas garante a segurança e a integridade dos dados contra falhas locais mas também facilita o acesso remoto e em tempo real pelos pesquisadores. Além disso, a infraestrutura do Firebase permite a implementação de triggers automáticos que podem ativar alertas ou ações baseadas em condições específicas detectadas nos dados.
4. Gestão de Conexões e Respostas: Com capacidades de comunicação sem fio, a placa utiliza protocolos como HTTP/1.1 para gerenciar as solicitações de dados via GET e POST, esta funcionalidade é essencial para permitir a integração com sistemas de monitoramento remoto e aplicativos móveis, tornando possível a visualização e análise dos dados em diferentes plataformas e dispositivos.

Este ciclo operacional garante que a placa BitDogLab opere de maneira autônoma e eficiente, proporcionando uma plataforma para o monitoramento ambiental das colmeias e oferecendo insights valiosos para os estudos de comportamento e saúde das abelhas.

5.3 Arquitetura

O desenvolvimento do software da placa começou com a escolha da linguagem Micropython para a programação. Micropython é uma implementação do Python 3 para microcontroladores, que abstrai funcionalidades de baixo nível usando a sintaxe de alto nível do Python. Essa escolha foi motivada pela familiaridade com a linguagem Python tradicional, além do

fato de ser base do projeto anterior, ponto importante, dado a continuidade dos trabalhos. Esse desenvolvimento estava disposto

O projeto do ano passado seguiu o paradigma da Programação Orientada a Objetos, utilizando uma arquitetura baseada em classes gerenciadoras para dividir suas funcionalidades principais em quatro classes:

- SensorsManager: Responsável pela leitura dos sensores.
- DatabaseManager: Gerencia o armazenamento dos dados.
- ServerManager: Administra a comunicação via internet.
- ConnectionsManager: Gerencia a conexão da placa à internet e/ou via Bluetooth.

A Figura 3 mostra uma esquematização das classes existentes anteriormente e suas relações.

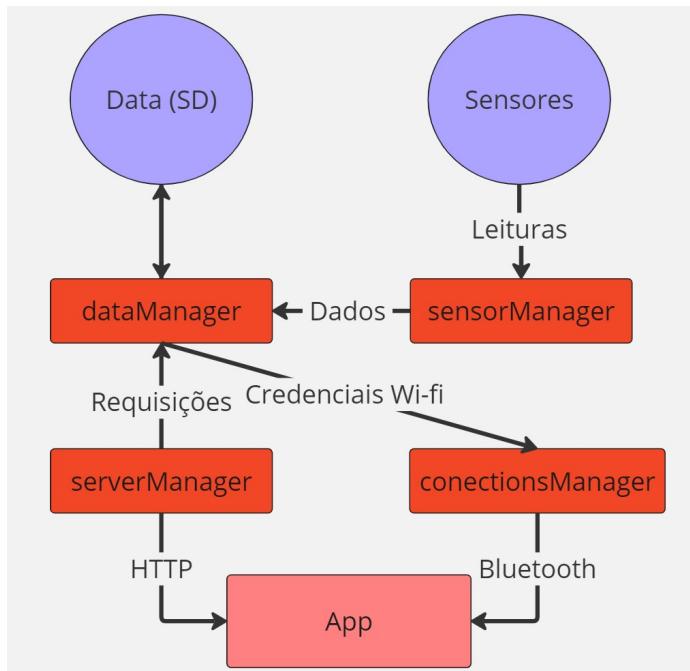


Figura 3: Diagrama da arquitetura do funcionamento anterior

A arquitetura da figura 3 é funcional para a solução antiga, em que a placa dependia do cartão SD. Porém, com o objetivo de utilizar o Firebase como centralizador dos dados e não necessitar mais do cartão SD, as classes e suas relações foram modificadas.

A Figura 12 a seguir mostra a nova relação entre as partes do projeto.

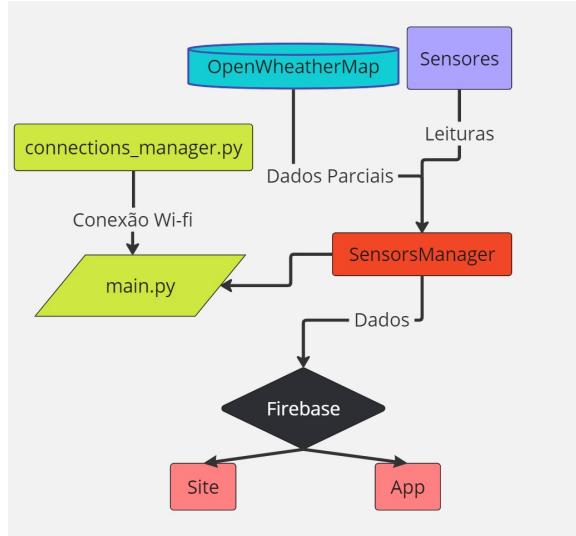


Figura 4: Diagrama atual da relação entre as partes

De modo geral, as partes relacionadas ao cartão SD e à conexão bluetooth foram completamente removidas.

5.4 Bibliotecas Externas e APIs

Além das bibliotecas padrão do MicroPython, foram incorporadas três bibliotecas e APIs específicas ao projeto, armazenadas na pasta *lib*:

- *open_weather*, utilizada para requisitar informações climáticas baseadas em coordenadas de latitude e longitude [25].
- *Firebase*, empregada para o envio e a requisição de dados através da ferramenta Real-time Database [21].
- *bme680*, utilizada para a leitura de dados do sensor BME680. Esta biblioteca é um port do MicroPython adaptado de uma biblioteca originalmente desenvolvida pela Adafruit para CircuitPython [22].

Todas as outras bibliotecas mencionadas nesta seção são padrão do MicroPython.

5.5 Estrutura do Software

A arquitetura do projeto é centralizada em torno da classe **SensorsManager**, que inicializa os sensores conectados à placa e realiza a leitura dos dados e envio para o armazenamento em nuvem.

Por padrão da Raspberry Pi, o script do arquivo **main.py** é executado quando a mesma é ligada a uma fonte de energia. O arquivo define e inicia a conexão Wi-Fi ao chamar a função `connect_to_wifi` do arquivo `connections_manager.py` e em seguida instancia a classe **SensorsManager**.

Uma mudança importante feita na versão atual do código é a de inicializar a leitura dos sensores com o método `sensors_reading()` da classe `SensorsManager` sem o uso da função `start_new_thread`, como foi feito no projeto do ano anterior.

Foi observado que isso permitiu que ao menos um sensor BME680 realizasse fosse iniciado sem haver o travamento do sistema assim que eram solicitados dados da API OpenWeatherMap ou enviados dados para o Firebase.

Ao utilizar a função `start_new_thread` ocorria o travamento em 100% das tentativas.

Os arquivos que foram incluídos na placa e podem ser consultados através da referência [18] são:

1. `main.py`: Inicia a conexão Wi-Fi e instancia a classe que administra os sensores e leituras.
2. `connections_manager.py`: Realiza a conexão Wi-Fi utilizando a biblioteca `network`.
3. `sensors_manager.py`: Contém a classe `SensorsManager`, com métodos para inicializar os sensores, realizar as leituras dos dados dos mesmos, requisitar dados complementares da API OpenWeatherMap e enviar a totalidade dos dados para o Firebase.
4. `OpenWeatherMap.py`: Contém os métodos para definir a localização, a URL, e requisitar os dados da API.
5. `http.py`: Contém a função `http_request`, que aplica o método GET da biblioteca `urequests` na url passada como parâmetro.
6. `uFirebase.py`: Contém as funções para ler e enviar dados para o Firebase utilizando GET, PUT e PATCH.

5.5.1 Leitura

Para realizar a leitura dos sensores, a classe `SensorsManager` precisa, primeiramente, instanciá-los, o que requer o conhecimento dos pinos aos quais cada sensor está conectado na Raspberry Pi. A pinagem de cada sensor, juntamente com seus valores na interface de entrada e saída do software (GPIO), é:

- Proximidade: pino 4, GPIO 2
- BME680 1: pinos 26 e 27, GPIO 20 e 21
- BME680 2: pinos 24 e 25, GPIO 24 e 25
- Microfone 1: pino 31, GPIO 26
- Microfone 2: pino 32, GPIO 27

A inicialização de cada sensor é armazenada nos atributos da instância de `SensorsManager`, de forma a evitar novas inicializações a cada ciclo de leitura.

Sensor de proximidade

O Sensor de Proximidade Ultrassônico E18-D80NK - NPN é um dispositivo utilizado para detecção de objetos sem contato físico, sendo ideal para aplicações em automação industrial, robótica, sistemas de segurança e controle de acesso. O datasheet do sensor E18-D80NK fornece todas as informações técnicas necessárias para a instalação, operação e manutenção do dispositivo. Abaixo estão os principais pontos abordados no datasheet do equipamento [23]:

- Modelo: E18-D80NK
- Dimensões: Diâmetro de 18mm, comprimento de 80mm
- Intervalo de Detecção: 3cm a 80cm
- Tensão de Operação: 5V-12V DC
- Corrente de Operação: <25mA
- Saída: NPN, normalmente aberto (NO)
- Frequência de Operação: 40kHz
- Material de Construção: Corpo em plástico ABS
- Temperatura de Operação: -25°C a 55°C

Os detalhes técnicos do sensor E18-D80NK incluem informações sobre seu funcionamento e características:

- Princípio de Funcionamento: O sensor utiliza ondas ultrassônicas para detectar objetos em sua proximidade. Quando um objeto é detectado, o sensor emite um sinal de saída através de seu transistor NPN.
- Calibração: O sensor não requer calibração frequente, mas pode ser ajustado para diferentes distâncias de detecção, se necessário.
- Manutenção: O sensor é praticamente livre de manutenção, necessitando apenas de limpeza periódica para remover poeira ou detritos que possam obstruir a emissão e recepção de ondas ultrassônicas.

Além disso ele é amplamente utilizado em várias aplicações devido à sua precisão e confiabilidade. Algumas das aplicações típicas incluem:

- Automação Industrial: Detecção de presença e posição de objetos em linhas de produção.
- Robótica: Navegação e prevenção de colisões em robôs móveis.
- Sistemas de Segurança: Controle de acesso e sistemas de alarme.

- Automação Residencial: Detecção de presença em portas automáticas e sistemas de iluminação.
- Controle de Nível: Medição de nível em tanques e reservatórios.

No projeto atual ele é usado para monitoramento de colmeias de abelhas, fornecendo dados para a gestão e manutenção das colônias, além do intuito pedagógico do projeto.

Ele pode ser instalado na entrada da colmeia para monitorar a atividade das abelhas, detectando o número de abelhas entrando e saindo, fornecendo informações sobre a saúde e a produtividade da colônia.

Com a emissão de ondas ultrassônicas e a detecção precisa de movimentos, o sensor registra cada passagem de abelha sem causar interferência ou estresse às mesmas.

Dentro da colmeia, o sensor pode ser utilizado para monitorar a posição das abelhas em áreas críticas, como nas proximidades da rainha ou em áreas de armazenamento de alimento, isso ajuda os apicultores a entenderem o comportamento das abelhas e a identificar possíveis problemas como congestionamento, falta de ventilação, obstruções ou qualquer outra anomalia presente na colmeia.

Como o único sensor que precisa ser monitorado continuamente é o de proximidade, o `timer` configurado no código encerra o loop de leitura após o tempo decorrido.

A leitura do sensor de proximidade, mostrado na figura ??, ocorre a cada 0,05 segundos. Isso é essencial para captar uma abelha que passa rapidamente, sem contar múltiplas vezes uma abelha que passa mais devagar. O sensor retorna 0 quando algo está em sua linha de detecção e 1 caso contrário. Esses valores são invertidos e adicionados a um atributo chamado `proximity_counter`, que registra quantas abelhas passaram pelo sensor a cada dia.

```
Scanning sensor
{'internal_pressure': 955.0337, 'internal_temperature': 23.95422, 'proximity': 940, 'internal_gas': 133116, 'internal_humidity': 66.73019}
READINGS {'internal_pressure': 955.0337, 'internal_temperature': 23.95422, 'proximity': 940, 'internal_gas': 133116, 'internal_humidity': 66.73019}
minutely 16.07.2024--22:22:22
```

Figura 5: Leitura do sensor de proximidade

Em cada leitura, a função `localtime` da biblioteca padrão `time` é chamada para obter o dia atual do mês, comparando-o com o valor armazenado em `current_day`. Se os dias forem diferentes, o loop de leitura de proximidade é interrompido independentemente do `timer`, permitindo que o último valor do contador diário seja salvo e, em seguida, zerado. O dia obtido pela função `localtime` é então atualizado em `current_day`.



Figura 6: Sensor de proximidade

Fonte: <https://www.usinainfo.com.br/sensor-de-proximidade/sensor-de-proximidade-e18-d80nk-infravermelho-npn-deteccao-3-a-80cm-2791.html>

Sensor sonoro

O Sensor GY-MAX4466 é um módulo de microfone de eletreto com amplificador embutido, ele é usado para capturar sons em uma ampla gama de aplicações, como gravação de voz, projetos de robótica, e sistemas de monitoramento de som. O datasheet do sensor GY-MAX4466 fornece todas as informações técnicas necessárias para a instalação, operação e manutenção do dispositivo. Abaixo estão os principais pontos abordados no datasheet[13]:

- Modelo: GY-MAX4466
- Dimensões: 9.7mm x 6.7mm x 4.5mm
- Tensão de Operação: 2.4V-5.5V DC
- Corrente de Operação: <0.85mA
- Saída: Analógica
- Resposta em Frequência: 20Hz a 20kHz
- Ganho do Amplificador: Ajustável (potenciômetro)
- Material de Construção: PCB de alta qualidade
- Temperatura de Operação: -40°C a 85°C

Os detalhes técnicos do sensor GY-MAX4466 incluem informações sobre seu funcionamento e características:

- Princípio de Funcionamento: O sensor utiliza um microfone de eletreto para captar sons, e um amplificador MAX4466 embutido para amplificar o sinal de áudio captado.
- Calibração: O sensor possui um potenciômetro para ajuste do ganho do amplificador, permitindo a calibração para diferentes níveis de som.
- Manutenção: O sensor é praticamente livre de manutenção, necessitando apenas de verificações periódicas para garantir que o potenciômetro esteja ajustado corretamente.

O sensor GY-MAX4466 é amplamente utilizado em várias aplicações. Algumas das aplicações típicas incluem:

- Gravação de Voz: Captação de áudio para gravação e processamento de voz.
- Sistemas de Monitoramento de Som: Monitoramento de ambientes para detecção de sons anômalos.

O sensor com sua capacidade de captação de sons ambientes, pode ser instalado dentro da colmeia para monitorar os sons produzidos pelas abelhas, isso inclui a detecção de zumbidos, batidas de asas e outras vocalizações que podem indicar o estado de saúde e atividade da colônia. Variáveis como frequência e intensidade dos sons podem fornecer informações sobre a presença da rainha, a saúde das abelhas operárias e a presença de possíveis ameaças. Alterações nos padrões sonoros dentro da colmeia podem indicar mudanças comportamentais importantes. Por exemplo, um aumento repentino na atividade sonora pode sinalizar que as abelhas estão em estado de alerta ou estresse, enquanto uma diminuição pode indicar um período de repouso ou um problema de saúde.

Além de monitorar sons dentro da colmeia, o sensor pode ser utilizado para captar sons no ambiente externo, isso pode ajudar a identificar fatores ambientais que afetam as abelhas, como ruídos de predadores, clima adverso ou atividades humanas próximas.

Sua leitura foi inicialmente realizada utilizando a biblioteca ADC, que converte o valor analógico do sensor em valores digitais entre 0 e 65535. Considerando que a capacidade máxima de leitura do sensor é de 60 dB, a equação utilizada para obter o valor em decibéis foi:

$$\text{Eq 1: } \text{valor_real} = 60 * \text{valor_lido} / 65535$$

Entretanto, a Figura 8a mostra a leitura dos sensores quando um deles foi exposto a um sopro e, posteriormente, a uma voz falada muito próxima. Como pode-se observar, não houve diferença significativa entre as duas situações, e os valores obtidos pareceram aleatórios. Para fins de comparação, a Figura 8b apresenta a mesma leitura com os sensores desconectados.

Um dos testes realizados foi a calibração dos sensores, que possuem um parafuso ajustável na parte traseira. Para um dos sensores, o parafuso foi girado ao máximo no sentido horário, enquanto para o outro, o parafuso foi girado no sentido anti-horário, repetindo-se o teste e obtendo-se os mesmos resultados. No entanto, valores diferentes foram observados quando o dedo do experimentador pressionou acidentalmente a parte traseira do sensor. Nessa condição, a leitura obtida era próxima de zero ou atingia o valor máximo, independentemente do som emitido. Isso indica que algo estava sendo detectado, embora não fosse o som desejado.

Durante a elaboração deste relatório, não foi possível fazer o sensor de som funcionar corretamente. Há poucas fontes online disponíveis para a leitura do MAX4466 usando Micropython, o que dificultou o trabalho e o tornou bastante experimental. O professor Fabiano sugeriu o uso de um sistema operacional em tempo real chamado NuttX, que ainda não foi testado, mas fica como uma sugestão de melhoria para tentativas futuras.



Figura 7: Sensor sonoro

Fonte: <https://www.smartkits.com.br/modulo-microfone-c-ganho-ajustavel-gy-max4466>

```

internal sound: 432
external sound: 416

internal sound: 37721
external sound: 36120

internal sound: 448
external sound: 432

internal sound: 34232
external sound: 34072

internal sound: 33496
external sound: 32904

internal sound: 35096
external sound: 33496

internal sound: 30935
external sound: 30951

```

(a) Sensores conectados

```

internal sound: 12899
external sound: 13331

internal sound: 13123
external sound: 13475

internal sound: 13971
external sound: 13923

internal sound: 14227
external sound: 14259

internal sound: 14275
external sound: 14339

internal sound: 13539
external sound: 13651

internal sound: 12403
external sound: 12675

```

(b) Sensores desconectados

Figura 8: Leitura dos sensores sonoros conectados(a) e desconectados(b)

Sensor BME680

O Sensor BME680 é um dispositivo multifuncional que combina sensores de gás, umidade, pressão e temperatura em um único pacote. Ele é ideal para aplicações que exigem monitoramento preciso do ambiente, como dispositivos de monitoramento de qualidade do ar, sistemas de automação residencial e wearable devices. O datasheet do sensor fornece todas as informações técnicas necessárias para a instalação, operação e manutenção do dispositivo. Abaixo estão os principais pontos abordados no datasheet[14]:

- Modelo: BME680
- Fabricante: Bosch Sensortec
- Dimensões: 3.0mm x 3.0mm x 0.93mm
- Tensão de Operação: 1.71V-3.6V DC
- Corrente de Operação: 0.15mA a 0.7mA (modo normal)
- Saída: I2C, SPI

- Faixa de Temperatura: -40°C a 85°C
- Faixa de Umidade: 0% a 100% RH
- Faixa de Pressão: 300hPa a 1100hPa
- Material de Construção: Encapsulamento em LGA de 8 pinos

Os detalhes técnicos do sensor BME680 incluem informações sobre seu funcionamento e características especiais:

- Princípio de Funcionamento: O sensor BME680 combina sensores de gás, umidade, pressão e temperatura para fornecer uma leitura abrangente da qualidade do ar e das condições ambientais.
- Calibração: O sensor vem calibrado de fábrica, mas pode ser ajustado através de software para atender a requisitos específicos de aplicação.
- Manutenção: O sensor é praticamente livre de manutenção, necessitando apenas de verificação periódica para garantir a precisão das medições.

O sensor BME680 é amplamente utilizado em várias aplicações. Algumas das aplicações típicas incluem:

- Dispositivos de Saúde: Monitoramento das condições ambientais em dispositivos médicos portáteis para garantir o conforto e a segurança do paciente.
- Sistemas de Climatização: Controle de sistemas HVAC (aquecimento, ventilação e ar condicionado) com base em medições precisas de temperatura, umidade e pressão.

A temperatura interna da colmeia é um parâmetro crítico para a saúde das abelhas, e o sensor permite monitorá-la com precisão, ajudando a garantir que a colmeia esteja dentro da faixa de temperatura ideal para o desenvolvimento das abelhas e a produção de mel. A umidade relativa dentro da colmeia também é um fator importante para a saúde das abelhas. Níveis inadequados de umidade podem levar ao crescimento de fungos e outras doenças.

Embora menos crítico que temperatura e umidade, a pressão atmosférica pode influenciar o comportamento das abelhas. Mudanças bruscas na pressão podem indicar mudanças climáticas que afetarão as atividades das abelhas.

A utilização do Sensor BME680 em colmeias de abelhas oferece uma solução fácil e barata para o monitoramento contínuo das condições ambientais, ajudando a fornecer dados que possam garantir a saúde e a produtividade das abelhas através de dados ambientais detalhados e precisos.

São necessários dois pinos para a leitura do sensor BME680, ilustrado na Figura 10. Isso ocorre devido ao uso do barramento I2C, que requer uma linha para dados (SDA) e outra para o clock (SCL).

Três bibliotecas foram encontradas para a leitura deste sensor, mas apenas a mencionada na Seção 5.4 retornou valores satisfatórios. Esta biblioteca é uma adaptação da versão

desenvolvida pela Adafruit para CircuitPython, criada em 2020 e aparentemente mantida por um único desenvolvedor. Portanto, é esperado que esteja em progresso e que nem todas as funcionalidades estejam completamente implementadas.

De fato, não é incomum que a instanciação da classe de leitura do BME680 resulte em erros, o que pode comprometer outras funções da placa. Quando esta classe está instanciada, a ativação do Bluetooth frequentemente resulta em timeouts, assim como a conexão WiFi. A leitura do banco de dados também se torna consideravelmente mais lenta, a ponto de se tornar impraticável. Por essas razões, apenas um sensor BME680 pôde ser inicializado enquanto as outras funções da placa no projeto eram mantidas funcionando corretamente.

```
Scanning I2C bus
('internal_pressure': 955.0337, 'internal_temperature': 23.95422, 'proximity': 940, 'internal_gas': 133116, 'internal_humidity': 66.73019)
READINGS {'internal_pressure': 955.0337, 'internal_temperature': 23.95422, 'proximity': 940, 'internal_gas': 133116, 'internal_humidity': 66.73019}
minimontain 16.07.2024-02:22:22
```

Figura 9: Leitura do sensor bme680

Apesar de os participantes de etapas anteriores do projeto considerarem causas ligadas ao BME680 como o problema, a situação seria resolvida com um micro-computador com mais poder de processamento e memória RAM que a Raspberry Pi Pico W.



Figura 10: Sensor BME680

Fonte: https://produto.mercadolivre.com.br/MLB-1591094884-sensor-de-gas-temperatura-presso-e-umidade-bme680-bme-680-_JM

API OpenWeatherMap

A API OpenWeatherMap oferece acesso a dados meteorológicos globais, permitindo que desenvolvedores integrem informações sobre clima em suas aplicações. Ela fornece dados em tempo real ou previsões sobre temperatura, umidade, precipitação, vento e outras condições meteorológicas. É ideal para aplicações que requerem dados meteorológicos precisos e atualizados. Os detalhes técnicos da API OpenWeatherMap incluem informações sobre endpoints, métodos de solicitação, formatos de resposta e autenticação[25].

Os principais endpoints da API são:

- Dados meteorológicos atuais: <http://api.OpenWeatherMap.org/data/2.5/weather>
- Previsão de 5 dias / 3 horas: <http://api.OpenWeatherMap.org/data/2.5/forecast>
- 16 dias / Previsão diária: <http://api.OpenWeatherMap.org/data/2.5/forecast/daily>

- Dados meteorológicos históricos: <http://api.OpenWeatherMap.org/data/2.5/timemachine>
- Dados de poluição do ar: http://api.OpenWeatherMap.org/data/2.5/air_pollution

A API OpenWeatherMap oferece uma variedade de funcionalidades, incluindo a obtenção de dados meteorológicos atuais para qualquer localização no mundo, esses dados incluem temperatura, umidade, pressão, velocidade e direção do vento, nebulosidade e condição climática. Além disso, a API fornece previsões meteorológicas de curto e longo prazo. A previsão de 5 dias/3 horas oferece detalhes a cada 3 horas, enquanto a previsão diária fornece uma visão geral dos próximos 16 dias. A API também permite acessar dados meteorológicos históricos, o que é útil para análises e estudos de tendências climáticas. Além disso, oferece dados sobre a qualidade do ar, incluindo níveis de poluição por PM2.5, PM10, NO2, SO2, O3 e CO, além de um índice geral de qualidade do ar.

Para solicitar dados meteorológicos, utiliza-se o método GET, que retorna um JSON contendo as informações requisitadas e para o acesso à API OpenWeatherMap é necessário obter uma chave de API (API Key). Essa chave deve ser incluída em todas as requisições para autenticação, como no exemplo abaixo:

```
http://api.OpenWeatherMap.org/data/2.5/weather?q=London&appid=YOUR\_API\_KEY
```

A resposta incluiria informações detalhadas, como coordenadas, condições meteorológicas, temperatura, pressão, umidade, visibilidade, velocidade e direção do vento, cobertura de nuvens, data e hora do relatório, informações sobre o sistema, identificação da cidade, nome da cidade e código de resposta. Abaixo está um exemplo de resposta para dados meteorológicos atuais:

```
{
  "coord": {"lon": -0.1257, "lat": 51.5085},
  "weather": [
    {"id": 800, "main": "Clear", "description": "clear sky", "icon": "01d"}
  ],
  "base": "stations",
  "main": {
    "temp": 280.32,
    "feels_like": 278.59,
    "temp_min": 279.15,
    "temp_max": 281.15,
    "pressure": 1012,
    "humidity": 81
  },
  "visibility": 10000,
  "wind": {"speed": 4.12, "deg": 80},
  "clouds": {"all": 0},
  "dt": 1605182400,
  "sys": {
```

```

    "type": 1,
    "id": 1414,
    "country": "GB",
    "sunrise": 1605152400,
    "sunset": 1605184800
},
"timezone": 0,
"id": 2643743,
"name": "London",
"cod": 200
}

```

5.5.2 Armazenamento em Nuvem

O armazenamento em nuvem é um componente crucial para o monitoramento remoto das colmeias, permitindo a coleta e o acesso aos dados em tempo real. Para este projeto, utilizamos o *Firebase Realtime Database*, uma solução de banco de dados em tempo real fornecida pelo Google.

O *Firebase Realtime Database* permite a sincronização dos dados entre os dispositivos em tempo real, garantindo que todas as leituras dos sensores sejam imediatamente acessíveis. A estrutura dos dados no *Firebase* é organizada em um formato de árvore JSON, onde cada nó representa um conjunto de dados de uma leitura específica, identificado por um carimbo de data/hora.

Na Figura 11, é possível observar a estrutura dos dados armazenados no *Firebase Realtime Database*. Cada entrada sob o nó `bee_data` contém as leituras dos sensores de temperatura e umidade, identificadas por um timestamp. Por exemplo, a entrada `06_05_2024-20_04_46` representa uma leitura realizada em 6 de maio de 2024 às 20:04:46, onde a temperatura registrada foi 1 e a umidade foi 2, a título de exemplo.



Figura 11: Estrutura de dados no Firebase Realtime Database

Para utilizar o Firebase é necessário possuir uma conta no mesmo. A criação da conta é realizada com os seguintes passos:

1. Acessar o site <https://firebase.google.com/>.
2. Clicar em "Get Started" e fazer login com uma conta Google.
3. Após o login, há um redirecionamento para o console do Firebase.

Com a conta criada, para realizar a criação de um novo projeto no Firebase é necessário seguir o passo-a-passo:

1. No console do Firebase, deve-se clicar em "Add project".
2. Inserir um nome para o projeto e clicar em "Continue".
3. Opcionalmente, ativar o Google Analytics para o projeto. Selecionar a opção desejada e clicar em "Continue".
4. Seguir as instruções na tela e clicar em "Create project".

5. Aguardar a criação do projeto e clicar em "Continue" para acessar o painel do projeto.

Para ativar o Realtime Database no projeto Firebase:

1. No painel do projeto, clicar em "Realtime Database" no menu lateral esquerdo.
2. Clicar em "Create Database".
3. Selecionar o modo de segurança. Para fins de desenvolvimento, escolher "Start in test mode". Para produção, selecionar "Start in locked mode" e configurar as regras de segurança posteriormente.
4. Escolher a localização do banco de dados (geralmente, a região mais próxima dos usuários) e clicar em "Done".

As regras de segurança definem quem pode acessar e modificar os dados no Realtime Database. Um exemplo básico de configuração de regras é apresentado abaixo:

```
{
  "rules": {
    ".read": "auth != null",
    ".write": "auth != null"
  }
}
```

- `.read`: Define quem pode ler os dados.
- `.write`: Define quem pode escrever os dados.
- No exemplo acima, apenas usuários autenticados podem ler e escrever no banco de dados.

Para adicionar o Firebase ao aplicativo, é necessário seguir as instruções específicas para cada plataforma:

- Web:
 1. No painel do projeto, clicar em "Add app" e selecionar "Web".
 2. Inserir um nome para o aplicativo e clique em "Register app".
 3. Seguir as instruções para adicionar o Firebase SDK ao projeto web.
 4. Copiar o snippet de configuração fornecido e colar no projeto.

Após configurar o Firebase no aplicativo, é possível começar a interagir com o Realtime Database, a implementação da funcionalidade de armazenamento e sincronização dos dados dos sensores no *Firebase Realtime Database* pode ser realizada utilizando a biblioteca `Firebase-admin` para Python ou a `Firebase` para MicroPython. O código abaixo exemplifica como os dados são estruturados e armazenados no `Firebase-admin`:

```

import Firebase_admin
from Firebase_admin import credentials, db

# Inicialização do Firebase Admin SDK
cred = credentials.Certificate('key.json')
Firebase_admin.initialize_app(cred, {
    'databaseURL': 'https://bee-rasp-default.firebaseio.com/',
})

# Referência ao nó principal
ref = db.reference('bee_data')

# Função para salvar dados no Firebase
def salvar_dados(timestamp, OpenWeatherMap, temperatura, pressao, proximidade,
                 som, umidade):
    ref.child(timestamp).set({
        'OpenWeatherMap': OpenWeatherMap,
        'temperatura': temperatura,
        'pressao': pressao,
        'proximidade': proximidade,
        'som': som,
        'umidade': umidade
    })

```

Este código inicializa a aplicação Firebase, estabelecendo a conexão com o banco de dados em tempo real. A função `salvar_dados` é responsável por armazenar as leituras dos sensores no nó `bee_data`, utilizando um carimbo de data/hora como identificador.

Agora abaixo temos um exemplo para Firebase do MicroPython:

```

import ujson
import urequests

Firebase_url = 'https://bee-rasp-default.firebaseio.com/bee_data'

def read_data():
    response = urequests.get(Firebase_url + '.json')
    response.close()
    if response.status_code == 200:
        data = response.json()
        print('Data read from Firebase:', data)
        return data
    else:
        print('Failed to read data:', response.status_code)
        return None

```

```

def write_data(data):
    headers = {'Content-Type': 'application/json'}
    response = urequests.put(Firebase_url + '.json', data=ujson.dumps(data),
    headers=headers)
    response.close()
    if response.status_code == 200:
        print('Data written to Firebase successfully')
    else:
        print('Failed to write data:', response.status_code)

def patch_data(data):
    headers = {'Content-Type': 'application/json'}
    response = urequests.patch(Firebase_url + '.json', data=ujson.dumps(data),
    headers=headers)
    response.close()
    if response.status_code == 200:
        print('Data patched in Firebase successfully')
    else:
        print('Failed to patch data:', response.status_code)

```

O *Firebase Realtime Database* se mostrou uma ferramenta eficaz para o armazenamento e recuperação rápida dos dados das colmeias, facilitando o monitoramento e análise contínuos das condições internas das colmeias.

5.5.3 Resposta a requisições

Como mencionado na seção Comunicação, a comunicação era mista e feita principalmente por Wi-Fi. Para se conectar, havia dois métodos possíveis: se a conexão já tivesse sido feita anteriormente, as credenciais estariam armazenadas na memória e poderiam ser usadas; ou, se fosse a primeira conexão na rede, as credenciais teriam que ser enviadas por Bluetooth. No segundo caso, utilizava-se o Bluetooth Low Energy (BLE), que consumia menos energia do que o Bluetooth comum, algo necessário para uma placa que poderia ser alimentada por bateria. Ao ser iniciada, a placa procurava pelo arquivo *wifi_credentials.txt*. Caso esse arquivo não existisse, o Bluetooth era ativado e a placa esperava a chegada de mensagens contendo o nome e a senha da rede Wi-Fi à qual deveria se conectar. Uma única mensagem Bluetooth não conseguia transmitir o nome e a senha inteiros, o que exigia o uso de uma sequência de mensagens. Abaixo estão os diretórios do SD:

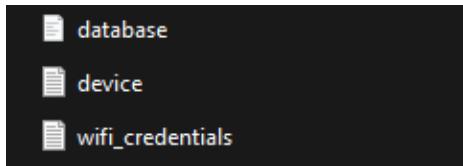


Figura 12: Arquivos dentro do cartãoSD

A conexão tinha 60 segundos para ser estabelecida. Se isso não ocorresse, assumia-se que a senha inserida estava incorreta. Nesse caso, a mensagem "not_connected" era enviada pelo BLE para o aplicativo, e aguardavam-se novas mensagens com outras credenciais. Caso a conexão fosse bem-sucedida, o IP obtido era enviado ao aplicativo via BLE, as credenciais recebidas eram enviadas ao *database_manager* para serem salvas e um socket na porta 80 era aberto. Esse socket era então passado ao ServerManager, que cuidava da conexão a partir desse ponto.

Além disso, o código contava com uma comunicação HTTP direta com o APP, assim era necessária uma requisição viva dos dados e a certeza de operação da placa 100% do tempo. Havia uma interdependência entre os pontos de informação e uma exigência do módulo de cartão SD e bluetooth serem utilizados, o que não é desejável.

O uso do bluetooth era mais um fator consumindo recursos do Raspberry Pi, atrapalhando o funcionamento do sensor BME680.

A solução foi simplificar o funcionamento do projeto mantendo uma credencial de Wi-fi fixa e uma taxa de envio de dados fixa, além do armazenamento dos dados no Firebase. Esses dois pontos dispensaram a necessidade de uso bluetooth e do micro SD, respectivamente, e o uso de threads adicionais. Isso liberou os recursos para o funcionamento com sucesso de um sensor BME680 e das requisições à API OpenWeatherMap, o que garantiu grandes avanços no cerne do projeto, que é a variedade de métricas obtidas.

Essa conexão simplificada e armazenamento no Firebase está em conformidade com o uso da radiofrequência LoRA e do multicast em próximos passos.

6 Aplicativo

De forma a exibir os dados provenientes de cada colmeia, a ideia inicial foi criar um aplicativo, como foi feito em anos anteriores.

- 2022: Foi desenvolvido um aplicativo Android que se comunicava apenas via Bluetooth e limitava a interação a uma única placa. Dado que a única funcionalidade reutilizável era a visualização de gráficos.
- 2023: Foi desenvolvido um novo aplicativo em Flutter, focado em utilização no sistema Android, realizando requisições HTTP e cadastro via Bluetooth, aproveitando a visualização de gráficos do código desenvolvido em 2022.

A ideia para o desenvolvimento atual seria aproveitar o possível do que foi feito anteriormente e adaptar o aplicativo para a arquitetura de componentes atual, em que não há

inicialização da placa através do mesmo e nem requisição de dados através de HTTP e da conexão Wi-FI. Dessa forma, o papel do aplicativo seria apenas exibir os dados requisitados do armazenamento em nuvem do Firebase.

6.1 Tecnologia aplicativo

Como decidiu-se manter a estrutura do trabalho passado, a framework de programação foi escolhido a Flutter[5], uma ferramenta open source desenvolvida pela Google, destinada a criar aplicações nativas para diversas plataformas a partir de um único código. A linguagem usada foi Dart[1].

6.2 Dados recebidos

As informações que se encontram no Firebase seriam recebidas pelo aplicativo na forma apresentada na figura.

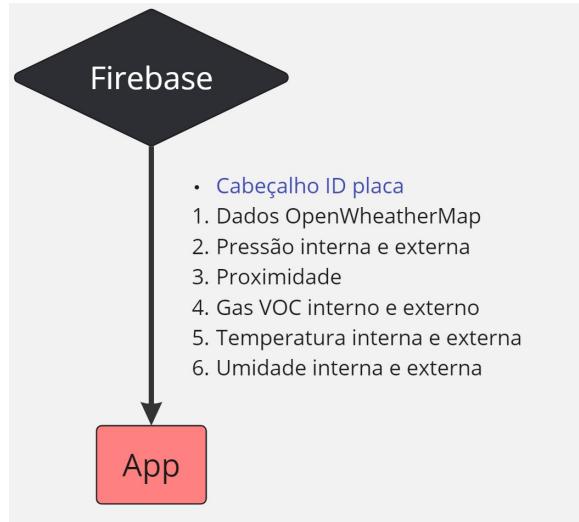


Figura 13: Estrutura de recebimento dos dados

6.3 Camadas de arquitetura

A estrutura do aplicativo no trabalho anterior era organizada em funcionalidades, como páginas de dados e dispositivos. Para manter características semelhantes entre diferentes funcionalidades, foi implementada uma organização em camadas, divididas em dados (data), lógica de negócios (business logic) e apresentação (presentation). Essa estrutura segue os princípios do padrão Bloc[12], visando maximizar a separação entre os dados e a interface do usuário, com a lógica de negócios funcionando como a ponte entre eles.

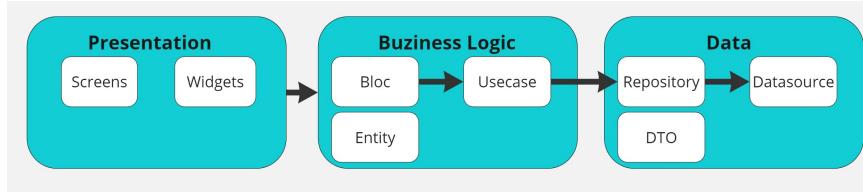


Figura 14: Diagrama sinalizando a comunicação entre as camadas de arquitetura do aplicativo

Essa estrutura para o projeto atual se manteve a mesma, o que difere é apenas como está organizado o bloco de dados. Nele se eliminará totalmente os métodos de bluetooth do Data Source, bem como se alterará as conexões em Repository de forma a todas se unificarem em métodos para atender ao consumo de dados do Firebase. A função de cada parte será:

- DTO (Data Transfer Object): Um Objeto de transporte de dados específicos entre o server e a aplicação. A ideia é tratar os dados Json usando a biblioteca `json_serializable`[6].
- Data Source: A classe tem por objetivo fazer a conexão entre o serviço e a DTO, a diferença será a de que todos os métodos devem ser voltados ao consumo de informação vinda pelo Firebase.
- Repository: Essa camada deverá ter o repositório de comunicação HTTP com o Firebase. Portanto, deve-se substituir a conexão direta e a conexão via bluetooth por uma referência ao Firebase.

Apesar dos esforços empreendidos em analisar a estrutura do aplicativo e o que seria necessário para adaptá-lo ao contexto atual, concluiu-se que uma de suas funções principais, a definição inicial de placas através do Bluetooth, não seria mais utilizada, e que teria uma manutenção mais custosa sem benefícios adicionais comparado a outras soluções.

O aplicativo também é burocrático no sentido de necessitar instalação prévia e ser compatível com um número limitado de dispositivos.

Assim, optou-se pela criação de um site para a visualização dos dados já armazenados no Firebase de forma acessível, simplificada e em tempo real.

7 WebApp

Como já mencionado, esse projeto conta com WebApp para visualização e interação do usuário com os dados coletados pela placa.

O projeto desenvolvido em 2023[26] contava com um aplicativo desenvolvido em flutter, porém este aplicativo era focado na configuração da placa via bluetooth e na visualização dos gráficos. Como foi realizado uma integração com o banco de dados Realtime Database do Firebase diretamente no Raspberry Pi, não havia mais a necessidade de configurar via Bluetooth a comunicação com a placa para obter os dados, sendo assim, o uso do aplicativo já não fazia mais sentido, pois sua única tarefa seria a visualização dos dados.

O site foi desenvolvido em React Js, conhecido por ser um Framework JavaScript muito flexível, completo e fácil de usar. Além de possibilitar a fácil visualização dos dados através de um site que está hospedado no próprio Firebase[19].

Essa adoção foi embasada no fato de ser mais fácil manipular os dados e termos diversas bibliotecas disponíveis para o uso. Pode-se destacar ainda que é possível fazer um aplicativo do tipo PWA[9] para ser utilizado por smartphones, não havendo a necessidade de usar uma linguagem ou Framework específico para mobile ou cross-plataform.

7.1 Objetivos do WebApp

O principal objetivo do WebApp é facilitar a visualização das datas dos dados possibilitando o usuário a interagir com cada dado de forma unitária, ou seja, escolher quais dados serão visualizados no gráfico

7.2 Tecnologia

A tecnologia escolhida para a elaboração do site foi o ReactJs, uma ferramenta desenvolvida pelo Facebook em 2011[7], o qual tornou seu código aberto em 2013. Uma das principais vantagens do React é sua abordagem baseada em componentes. Componentes são blocos de construção independentes e reutilizáveis que podem ser combinados para criar interfaces de usuário complexas. Essa modularidade não apenas facilita a manutenção do código, mas também promove a reutilização de componentes em diferentes partes da aplicação, aumentando a eficiência e velocidade do desenvolvimento.

Outro ponto importante, o React utiliza uma versão virtual da DOM (Document Object Model)[2] para otimizar as atualizações de interface do usuário, e também, suporte para SSR (Server-Side Rendering). O virtual DOM permite que o React realize mudanças de maneira mais eficiente, atualizando apenas os elementos que realmente sofreram alterações. Isso resulta em uma performance melhor e, mais importante, em uma experiência do usuário muito fluida, não havendo a necessidade de recarregar a página a todo o momento.

Essa característica será fundamental para a visualização em tempo real dos dados da placa, que será detalhado na seção de obtenção dos dados. Já o SSR, permite renderizar a página no servidor antes de ser enviada ao cliente, proporcionando um tempo de carregamento menor, especialmente em conexões lentas.

Por fim, ao possuir um ecossistema vasto e uma comunidade extremamente ativa, foi possível utilizar componentes já prontos para exibir o gráfico, manter a conexão com o Firebase e entre outros.

Apesar de existirem outros Frameworks muito bons, como Angular, sua estrutura básica é um pouco mais complexa. Se por um lado, isso ajuda a manter um padrão bem definido desde o início do desenvolvimento, por outro pode afastar desenvolvedores menos experientes e fazer com que o desenvolvimento da aplicação seja um pouco mais demorado. Sendo assim, foi optado pelo React devido à sua baixa complexidade e uso/reuso de componentes. Vale ressaltar também, que com poucas configurações é possível adicionar a opção de PWA (Progressive Web App)[8] a um projeto em React, o qual pode ser instalado em qualquer

plataforma mobile (Android, iOS, Windows Phone) que possua acesso à internet, por usar o navegador como base de uso.

Vale ressaltar que um PWA não é instalado através de uma loja de aplicativo, ele roda em cima do navegador, porém um atalho é criado na tela inicial, semelhante a um aplicativo nativo. Ao abrir o PWA todas as características do navegador são omitidas, como a barra de navegação, sendo visualizado apenas o conteúdo da página web.

7.3 Estrutura e funcionamento WebApp

O WebApp consiste em uma tela com o gráfico interativo, sendo possível escolher quais dados enviados pela Raspberry devem ser mostrados.

Para melhor estruturação do código, foram criados dois componentes para a elaboração da aplicação. O componente principal possui a requisição para o banco de dados, banco que será detalhado em sua própria seção, e um outro componente para a inserção dos dados no gráfico.

No componente principal, foi utilizada a funcionalidade de `UseEffect()`. Essa é uma função muito interessante, pois é capaz de executar efeitos colaterais em componentes funcionais fora do ciclo de renderização do React, sendo ativada sempre que o valor de sua variável interna mudar. Dentro dela é realizada a requisição para o banco de dados através de uma função da própria biblioteca do Firebase, o método `OnValue()`, o qual recebe a referência do banco de dados e uma variável para retornar os valores(`snapshot`).

Ao realizar a requisição para a raiz do banco, recebe-se um `snapshot`, o qual possui todas as “chaves”, sendo essa chave um timestamp para identificação de cada coleta de dado. Cada chave possui dois outros objetos (“OpenWeatherMap” e “sensor”), sendo o primeiro o retorno da API OpenWeatherMap e o segundo as leituras dos sensores da Raspberry, ambos objetos enviados pela placa.

Deste `snapshot`, mapeamos cada valor de temperatura, umidade, pressão e sensação térmica para um array específico, sendo estes arrays os valores que o gráfico deverá exibir no eixo Y. Já o eixo X, receberá as chaves que são um timestamp do momento que a placa coletou aqueles dados. A modelagem será aprofundada na sua própria seção.

Quando a primeira requisição é realizada, a comunicação entre o banco de dados e o webapp é mantida aberta pelo método `OnValue()`. Assim, quando há uma alteração nos dados do banco, o Firebase envia a atualização para todos os clientes conectados. Ao receber esta atualização, como ela está dentro do `UseEffect()`, os valores dos dados enviados para o componente do gráfico são atualizados, renderizando apenas o componente do gráfico para o usuário e evitando de precisar passar por todo o ciclo de renderização da página completa.

8 Firebase

Para armazenamento de dados e hosting (hospedagem de um WebApp), foi optado pelo uso do Firebase, plataforma desenvolvida pela Google[3]. O Firebase é uma plataforma muito completa, que oferece diversas ferramentas e serviços para o desenvolvimento de aplicativos móveis e web.

Dentre suas ferramentas, podemos citar armazenamento de dados, autenticação, hospedagem de sites, analytics, entre outros. Analytics é uma ferramenta que possibilita a análise de uso do aplicativo (móvel ou web) e comportamento do usuário.

A escolha do Firebase leva em conta esses pontos mais fortes, além de possuir uma documentação extremamente detalhada, fácil integração e configuração através do uso do Firebase CLI[4] e, principalmente, um banco de dados em tempo real (Realtime Database).

8.1 Dados

O armazenamento dos dados é realizado através do serviço de Realtime Database[10], o qual usa um banco de dados do tipo NoSQL, mais especificamente do tipo documento. Apesar de bancos SQL serem mais performáticos, havia a necessidade de ser possível a visualização dos dados em tempo real, sendo esta função disponível apenas em bancos NoSQL. Em um banco SQL, haveria a necessidade de criar um sistema de pooling, ou seja, consultar o banco de dados constantemente para manter os dados no WebApp atualizados, ou utilizar um sistema de WebHook, no qual o banco de dados iria informar que houve uma alteração na tabela, podendo ou não, fornecer estes novos dados. Em ambos os casos, haveria um aumento na complexidade da aplicação, culminando na necessidade de dividir nossa aplicação em Front-end e Back-end, a fim de dividir melhor as responsabilidades.

Dessa forma, visando manter uma menor complexidade do sistema e atender as necessidades do projeto, o serviço de Realtime Database mostrou-se mais interessante.

8.2 Modelagem

Existem estruturas diferentes de banco de dados NoSQL, como documentos, chave-valor, de coluna ou de gráficos. Porém a estrutura de uso mais recorrente em bancos NoSQL é a estrutura de documentos. Ela é uma grande árvore JSON, sendo possível armazenar unidades de dados independentes, contendo informações como chave-valor, listas ou objetos aninhados. Outro ponto importante, o formato JSON é a principal estrutura de dados usada para envio e recebimento de dados, especialmente na comunicação entre cliente e servidor. Assim, caso haja necessidade de criar um Back-end dedicado no futuro, basta alterar o endereço de requisição, já que o mapeamento deste JSON pode ser reaproveitado.

Visando a fácil visualização e manipulação dos dados, a estrutura do banco foi modelada da seguinte forma como na figura.

```
{
  "bee_data": {
    "08_07_2024-23_37_23": {
      "openweathermap": {
        "base": "stations",
        "clouds": { ... },
        "cod": 200,
        "coord": { ... },
        "dt": 1721183824,
        "id": 3467865,
        "main": {
          "feels_like": 290.31,
          "grnd_level": 950,
          "humidity": 88,
          "pressure": 1025,
          "sea_level": 1025,
          "temp": 290.25,
          "temp_max": 291.07,
          "temp_min": 289.91
        },
        "name": "Campinas",
        "sys": { ... },
        "timezone": -10800,
        "visibility": 5000,
        "weather": [ ... ],
        "wind": { ... }
      },
      "sensor": {
        "gas_interno": 149844,
        "id_placa": 1,
        "pressao_interna": 955.074,
        "proximidade": 3602,
        "temperatura_interna": 23.95481,
        "umidade_interna": 66.62566
      }
    },
    "08_07_2024-23_37_44": { ... },
    "08_07_2024-23_38_04": [ ... ]
  }
}
```

Figura 15: Estrutura do banco de dados

Essa modelagem foi pensada, também, em manter um forte controle sobre o exato momento de leitura dos sensores e da API, garantindo que esses dados fiquem armazenados juntos e evitando duplicidade de valores dentro dos objetos que pertencem à mesma leitura. Além disso, o cliente pode facilmente escolher o intervalo de tempo que deseja observar os dados, não sendo necessário iterar por todos os objetos e agrupá-los pela data da leitura.

8.3 Hosting

Outro serviço utilizado para melhorar o uso do projeto, foi a hospedagem do WebApp através do Firebase, sendo possível a leitura dos dados em qualquer navegador, apenas acessando a URL. Este serviço é muito bem integrado ao ecossistema do Firebase, bastando configurar um arquivo “Firebase.json” na raiz do projeto e especificando qual pasta possui a build do projeto, neste caso na própria pasta build do React.

```
{
  "hosting": {
    "public": "build",
    "ignore": [
      "firebase.json",
      "**/.*",
      "**/node_modules/**"
    ],
    "rewrites": [
      {
        "source": "**",
        "destination": "/index.html"
      }
    ]
  }
}
```

Figura 16: Estrutura de dados do Firebase.json

Após a configuração, foi usado o comando “\$ Firebase deploy” do Firebase CLI, o qual realiza a hospedagem e disponibiliza a aplicação em dois subdomínios do próprio Firebase, nome-projeto.web.app e nome-projeto.firebaseioapp.com.

9 Documentação

A seguir, são listados os dois repositórios com os códigos gerados e outras referências para acesso ao conteúdo gerado.

- O repositório com os arquivos em MicroPython com os scripts que estão inseridos no microcontrolador para realizar a coleta de dados e envio dos mesmos para o armazenamento do Firebase pode ser acessado pela referência [18]. O repositório é publico e o código final e atualizado se encontra na branch ‘main’.
- O repositório contendo os arquivos que compõem o site em que os dados são exibidos pode ser acessado através da referência [17]. O repositório é publico e o código final e atualizado se encontra na branch ‘main’.
- O repositório que contém os arquivos inseridos na placa BitDogLab[11], incluindo firmwares, além de documentação, esquemáticos, libs e softwares extras para usos majoritariamente didáticos pode ser encontrado em [11].

- O documento referente à placa contendo suas especificidades é encontrado na referência [16].
- O vídeo de demonstração de uso do site que exibe os dados que estão armazenados no Firebase é encontrada na referência [20].

10 Resultados

Os resultado obtidos foram:

- Coleta de dados de proximidade, temperatura, umidade, pressão e gases voláteis através dos sensores conectados à placa.
- Coleta de dados através da API da plataforma OpenWeatherMap.
- Armazenamento dos dados em um banco de dados hospedado em nuvem pelo Firebase.
- Criação de site hospedado pelo Firebase para fácil visualização dos dados coletados, com escolha de quais medições são exibidas.

É importante salientar que o grupo que trabalhou com o projeto no ano anterior teve dificuldades para atingir o funcionamento do sensor BME680 e do sensor de som GY-MAX4466. Isso se deveu ao alto consumo de recursos do micro-computador pelos mesmos, incompatível com o uso simultâneo de conexão bluetooth, que era necessária para conexão inicial da placa. Dessa forma, o único dado coletado com sucesso na ocasião era o do sensor de proximidade E18-D80NK.

Assim, um avanço importante foi o de que atualmente os dados de um sensor BME680 são coletados e exibidos com sucesso, aumentando significativamente a variedade de informações coletadas.

Um resultado esperado não atingido é o da coleta de dados com um segundo sensor BME680 em paralelo ao primeiro. O script para tal foi criado, porém não é possível inicializar dois sensores BME680 e realizar requests (no caso, requests ao OpenWeatherMap) ou realizar push dos dados para o Firebase simultaneamente. Isso se deve ao alto consumo de recursos do BME680 em relação ao processamento do Raspberry Pi Pico W.

A coleta de dados através da API da OpenWeatherMap foi algo sugerido na conversa inicial com o professor Roberto Greco e foi entregue como esperado. Os dados podem ser observados em conjunto com os obtidos pelos sensores conectados à placa, o que permite análises mais robustas.

Outro ponto que foi corrigido em relação ao projeto do ano anterior foi o armazenamento dos dados. Os mesmos eram armazenados no cartão SD inserido na placa, o que era um fator limitante considerando que estavam sendo guardados localmente, em contraste ao armazenamento confiável e de fácil acesso e manipulação, em nuvem do Firebase.

A visualização dos dados através do site [19], criado no decorrer desse semestre, é mais fácil, independente de plataforma e não necessita de download e instalação de aplicativo, como ocorria anteriormente.

A interface do site, exibindo os dados nos gráficos, pode ser vista nas imagens.

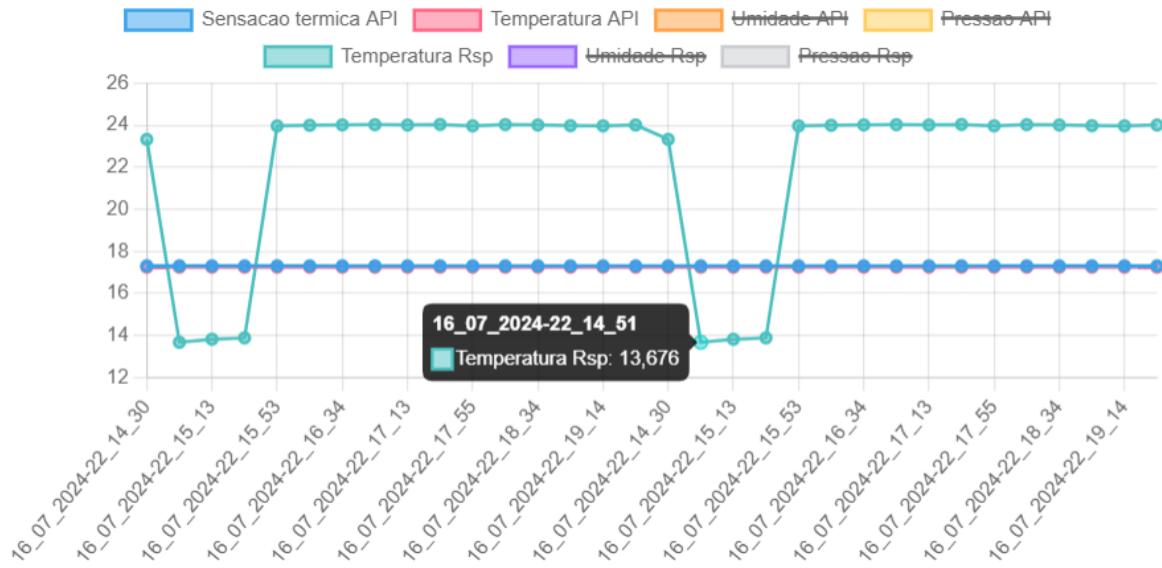


Figura 17: Gráfico da temperatura

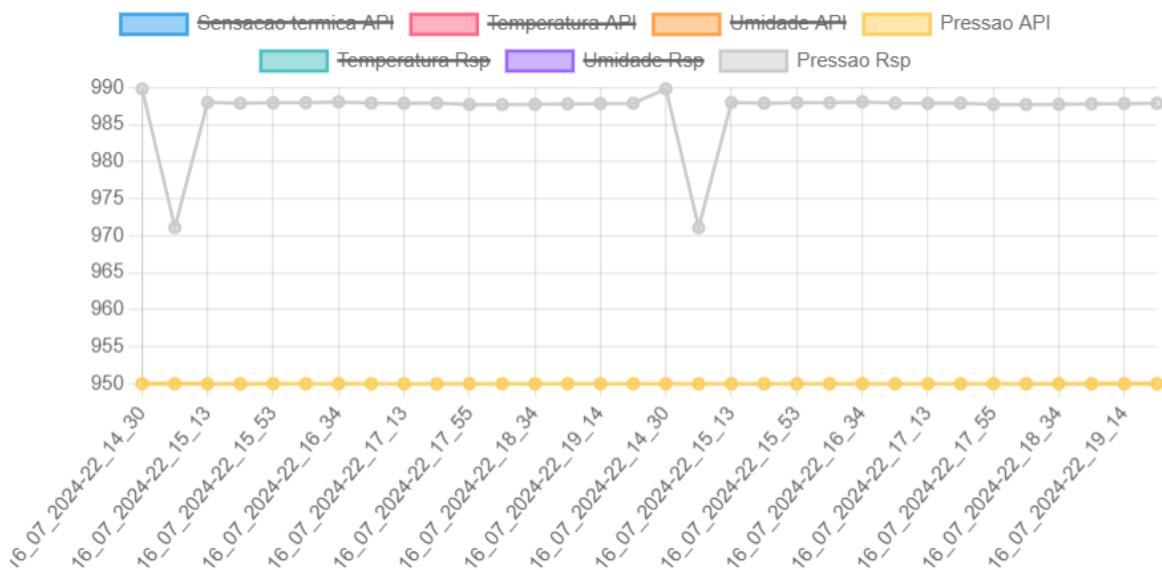


Figura 18: Gráfico da pressão

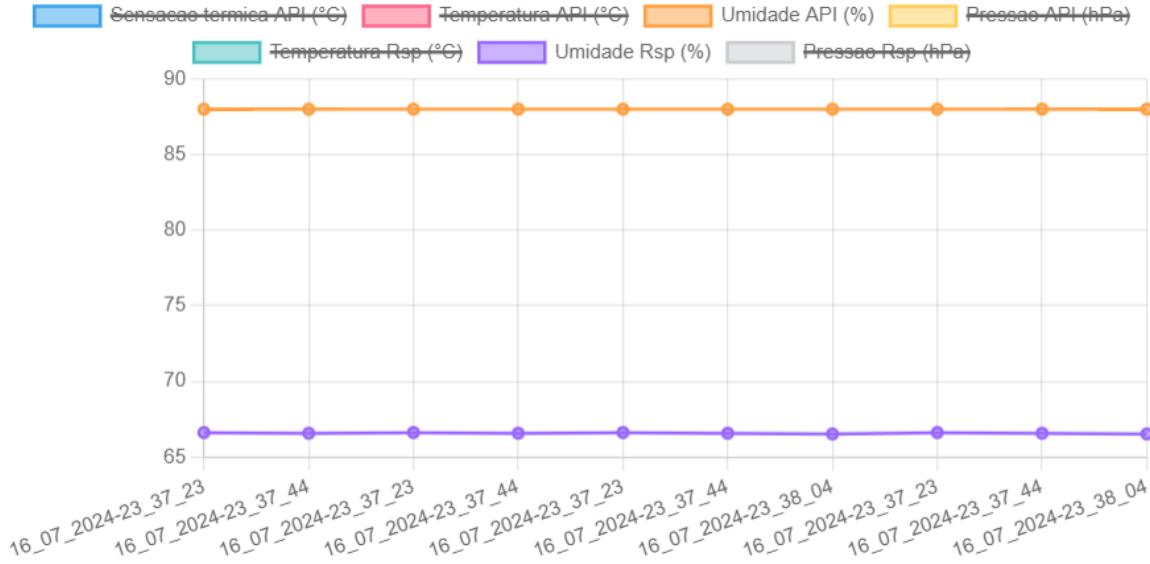


Figura 19: Gráfico da umidade

Os avanços obtidos nesse semestre foram, em suma, a maior variedade de dados coletados, armazenamento mais robusto e visualização mais fácil dos mesmos.

11 Modo de uso

O modo de uso foi simplificado. Para utilizar a placa, basta conectá-la à tomada com o nome da rede wi-fi e senha pré-escritos no script inserido na mesma.

Isso se deve ao fato de que, como trabalho futuro, as placas nas escolas se comunicarão através da radiofrequência LoRa, e, dessa forma, apenas as placas que servirão como gerenciadoras precisarão estar conectadas à rede wi-fi.

Para acessar os dados, basta acessar o link do site [19] no navegador e dispositivo de preferência. O usuário pode escolher os parâmetros que serão exibidos de acordo com as instruções do site. A demonstração de uso do site pode ser acessada através da referência [?].

12 Conclusão e Trabalhos Futuros

Diante do trabalhado ao longo do semestre, pode-se observar que muitos dos objetivos iniciais foram alcançados e, apesar dos muitos problemas enfrentados, o grupo pôde usufruir da experiência pedagógica proposta pelo PFG. Nesse sentido cabe destacar as seguintes conquistas:

- Funcionamento do sensor BME680 : Nos projetos anteriores, um dos maiores problemas encontrados era não ser possível a inicialização do sensor BME680 concomitantemente com a conexão bluetooth sem que o sistema parasse de funcionar. Após

a simplificação do código o sensor passou a funcionar corretamente e não conflita com a comunicação da placa.

- Comunicação com Firebase: Como a ideia do projeto é ter sua comunicação placa/aplicação independentemente da distância, a implementação do Firebase fez com que o projeto ganhasse uma maior independência entre a placa e os dispositivos de usuário pois, mesmo a placa deixando de funcionar por motivos externos, os dados captados ainda são consumíveis pela aplicação.
- Independência do SD : Além disso, removeu-se a interdependência entre aplicação e o cartão SD da placa. Aqui fixou-se uma identidade para a placa e removeu-se a necessidade do uso do bluetooth para que a placa sua inicialização, assim diminuindo o consumo de recursos da placa Raspberry Pi Pico W.
- BitDogLab: Com as mudanças e adaptações feitas, agora é possível usar a placa BitDogLab[11], em que mais pessoas poderão contribuir para o projeto além dos alunos da computação. O mais interessante é que será possível mesclar os projetos em um único projeto pedagógico. Criando então um vínculo multidisciplinar entre o trabalhado pelo professor Roberto e o professor Fabiano.
- Aplicação Web: Para facilitar o consumo e consulta dos dados, a aplicação web é importante para a que não se necessite uma constante atualização das arquivos .apk do aplicativo. Aumentou-se a simplicidade e a velocidade do consumo das informações.

Apesar das conquistas o grupo teve alguns objetivos que não foram completos. O principal deles foi a integração do antigo aplicativo à nova comunicação. Como o aplicativo estava muito dependente da antiga forma de comunicação, encontrou-se dificuldade para fazer com que o mesmo funcionasse adequadamente.

Ademais, algo que também se constatou como um problema foi a utilização de métodos HTTP para envio e recebimento de dados concomitantemente com o BME680. A solução envolveu a eliminação de threads desnecessárias e a eliminação de uso de bluetooth, liberando recursos para a inicialização de ao menos um sensor BME680. A conclusão do grupo foi que o micro-computador utilizado não consegue lidar com tantos processos simultaneamente, mesmo com a liberação ativa de RAM.

Dante disso, foi discutido alguns tópicos para serem resolvidos ou implementados em futuros trabalhos:

1. Estudo do sensor de áudio: Foi conversado com o professor Fabiano que o método de captação do áudio como está sendo feito atualmente não é eficiente, pois é necessário um tratamento extensivo na forma como são coletados. O professor propôs um futuro PFG voltado a um estudo dedicado a isso.
2. OpenWeatherMap independente da placa: Para que não se sobrecarregue a placa, propõe-se que futuramente a OpenWeatherMap esteja implementada diretamente nas aplicações, assim, liberando recursos da placa e aumentando a independência das aplicações.

3. Integrar a comunicação de rádio LoRa: Após uma discussão com o grupo, foi comentado que grande parte das colmeias não fica em áreas com acesso Wi-Fi. Uma das motivações da troca de placa feita pelo grupo foi a capacidade da placa BitDogLab de suportar a comunicação via ondas de radiofrequência LoRa, que permitem conexão em alcances substancialmente maiores.

Isso seria especialmente útil na implementação do multicast, com diversas placas realizando coletas simultaneamente e se comunicando com uma placa gerenciadora desses dados, que faria o envio para o Firebase, onde os mesmos seriam consumidos pelo site.



Figura 20: Arquitetura proposta pro futuro

Referências

- [1] *Dart*. Disponível em: <https://dart.dev/>.
- [2] *Dom*. Disponível em: <https://pt-br.legacy.reactjs.org/docs/faq-internals.html>.
- [3] *Firebase*. Disponível em: <https://firebase.google.com/>.
- [4] *Firebase cli*. Disponível em: <https://firebase.google.com/docs/cli>.
- [5] *Flutter*. Disponível em: <https://flutter.dev/>.

- [6] *google.dev: json_serializable.* Disponível em: https://pub.dev/packages/json_serializable.
- [7] *História do react js.* Disponível em: <https://cursa.com.br/blog/details/a-hist%C3%B3ria-do-react-js-uma-jornada-pioneira-na-web-moderna/21>.
- [8] *Making a pwa.* Disponível em: <https://create-react-app.dev/docs/making-a-progressive-web-app/>.
- [9] *Pwa.* Disponível em: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps.
- [10] *Realtime database.* Disponível em: <https://firebase.google.com/docs/database>.
- [11] *Repositório bitdoglab.* Disponível em: <https://github.com/BitDogLab/BitDogLab>.
- [12] *Why bloc?* Disponível em: <https://bloclibrary.dev/#/whybloc>.
- [13] ANALOG, *Datasheet sensor gy-max4466.* Disponível em: <https://www.analog.com/en/products/max4466.html>.
- [14] BOSCH, *Datasheet sensor bme680.* Disponível em: <https://cdn-shop.adafruit.com/product-files/3660/BME680.pdf>.
- [15] R. T. FIELDING, M. NOTTINGHAM, AND J. RESCHKE, *HTTP Semantics.* RFC 9110, June 2022.
- [16] F. FRUETT AND R. GRECO, *Multiple sensor beehive monitoring.* Disponível em: <https://docs.google.com/document/d/1Mz-MCR6UlPnqEdy0TakaD4TGH3IKm6izRvWRS6cQM8/edit>, Fevereiro 2023.
- [17] J. C. GONÇALVES, D. M. D. SANTOS, L. J. S. D. S. P. MONROE, AND V. A. SCHOLZE, *apicultores app.* Disponível em: <https://github.com/VictorScholze/apicultoresWeb>.
- [18] ——, *bee monitoring micropython.* Disponível em: https://github.com/JonasCardoso/bee_monitoring_BitDogLab.
- [19] ——, *Site para visualização dos dados.* Disponível em: <https://bee-rasp.firebaseioapp.com/>.
- [20] ——, *Vídeos demonstrativos do projeto.* Disponível em: <https://drive.google.com/file/d/1gA99jFXzEFxt4vWKGEL1m02-WT28STXf/view?usp=sharing>.
- [21] GOOGLE, *Firebase.* Disponível em: <https://firebase.google.com/docs/database?hl=pt-br>.
- [22] R. HAMMELRATH, *micropython driver for a bme680 breakout.* Disponível em: <https://github.com/robert-hh/BME680-Micropython>.

- [23] U. INFO, *Datasheet sensor e18-d80nk.* Disponível em: <https://www.usinainfo.com.br/sensor-de-proximidade-sensor-de-proximidade-e18-d80nk-infravermelho-npn-deteccao-3-a-80cm-2791.html>.
- [24] G. J. S. MORAES, V. A. M. DANTAS, A. C. L. C. C. F. RENOLDI, H. F. ZIMERMANN, P. P. ALVES, J. V. F. COSTA, L. S. L. CARMO, AND L. F. BITTENCOURT, *Monitoramento de colmeias com internet das coisas*, (2022).
- [25] OPENWEATHER, *Weather api.* Disponível em: <https://openweathermap.org/api>.
- [26] M. C. ROSA, L. C. CASTELLO, C. A. A. TRUJILLO, AND L. F. BITTENCOURT, *Monitoramento de colmeias com internet das coisas*, (2023).