Relatório de Projeto de Internet das Coisas (IoT)

SISTEMA DE MONITORAMENTO TÉRMICO INDEPENDENTE PARA CABINE DE VEÍCULOS AUTOMOTIVOS

Ernane Ferreira Rocha Junior
Quelita Míriam Nunes Ferraz

SISTEMA DE MONITORAMENTO TÉRMICO INDEPENDENTE PARA CABINE DE VEÍCULOS AUTOMOTIVOS

Disciplina: IMD0902 - Internet das Coisas - 2022.2

Orientador: Prof. Heitor Medeiros Florencio

Universidade Federal do Rio Grande do Norte

Descrição do projeto

A necessidade de monitorar a temperatura e a umidade dentro da cabine do veículo de forma independente surge da importância de garantir conforto e segurança para os ocupantes. Temperaturas extremas e níveis inadequados de umidade podem impactar negativamente a saúde, causando desconforto, fadiga ou até problemas respiratórios. Além disso, em veículos que transportam cargas sensíveis ou que são utilizados em climas variados, a monitoração precisa e independente desses parâmetros é crucial para manter um ambiente ideal e prevenir danos. Um sistema independente de monitoramento permite respostas rápidas e automação de ajustes, melhorando a eficiência e a qualidade da experiência dentro do veículo.

Resumo

Este projeto visa desenvolver e implementar um dispositivo IoT para monitoramento térmico independente dentro da cabine de veículos automotivos. Utilizando um sensor DHT11, o dispositivo coleta dados de temperatura e umidade, que são processados por um microcontrolador ESP32. Os dados são exibidos em tempo real em um display LCD e enviados para a nuvem via Wi-Fi, onde são armazenados e visualizados em dashboards. O sistema proporciona monitoramento contínuo das condições ambientais na cabine, garantindo conforto e segurança para os ocupantes.

A metodologia de desenvolvimento incluiu levantamento de requisitos, programação dos dispositivos, e prototipação inicial, resultando em uma solução eficiente e robusta. Abrange desde a concepção e montagem do dispositivo até os testes práticos realizados. Serão discutidos os componentes utilizados, o processo de integração no veículo, a configuração do sistema de comunicação com o broker da Adafruit, além dos resultados obtidos durante os testes de campo.

Ao longo do documento, serão apresentados detalhes técnicos importantes, considerações de projeto, desafios enfrentados e soluções implementadas para garantir o sucesso e a confiabilidade do sistema de monitoramento desenvolvido.

1. Introdução

O presente relatório descreve o desenvolvimento e implementação de um dispositivo IoT capaz de monitorar e registrar as variações térmicas dentro de um veículo automotivo. O projeto utiliza o microcontrolador ESP32 em conjunto com o sensor de temperatura e umidade DHT11 para capturar dados ambientais relevantes e um display LCD16x2 para exibição dos dados. Os dados coletados são então enviados para um servidor de nuvem (broker da Adafruit), onde são processados e disponibilizados para análise através de um painel de controle que exibe suas informações em tempo real incluindo, mas não se limitando à, gráficos das duas variáveis principais (umidade e temperatura) bem como um 'simulador' Termo Higrômetro digital.

O dispositivo foi concebido para ser integrado no interior de um veículo específico, neste caso, um GM Tracker 2007. O monitoramento térmico visa fornecer informações cruciais em duas principais áreas de interesse: primeiro, determinar o tempo necessário para que o veículo atinja sua temperatura máxima, qual será essa temperatura naquele dia em específico, quando estacionado ao sol em condições típicas da cidade de Parnamirim, Rio Grande do Norte e, por fim, qual a temperatura média obtida ao longo do processo. Segundo, avaliar o tempo que o sistema de ar-condicionado do veículo leva para alcançar a temperatura mínima, possível, após o veículo ter atingido sua temperatura máxima obtendo as mesmas métricas de média e qual a temperatura média obtida.

2. Arquitetura IoT

A arquitetura IoT do projeto integra sensor, microcontrolador, exibição local e armazenamento em nuvem de forma eficiente. Essa integração permite um monitoramento abrangente e em tempo real das condições ambientais da cabine, oferecendo benefícios significativos para a segurança e conforto dos ocupantes do veículo.

Por meio da figura abaixo podemos entender sobre as camadas do dispositivo. A “coisa” utilizada é o veículo automotivo, que deseja-se coletar dados de temperatura e umidade através do sensor DHT11, que envia seus dados para o ESP32. Tal microcontrolador atua como o cérebro do sistema, recebendo os dados do sensor DHT11 e realiza as seguintes funções: calcula as médias de temperatura e umidade e envia os dados processados para um display LCD para visualização imediata, como também para a nuvem para armazenamento e análise.

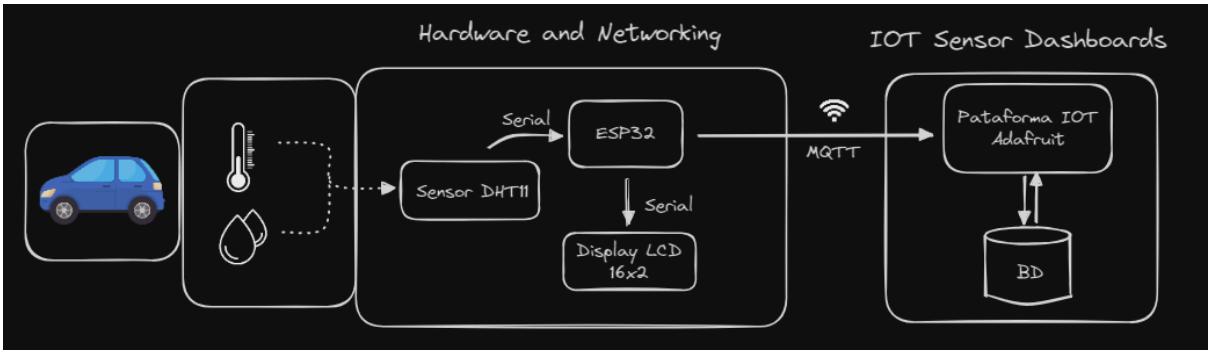


Figura 01 - Arquitetura do projeto. Fonte: Autores.

A plataforma IOT utilizada no projeto é o AdaFruit, que recebe os dados enviados pelo ESP32 e os armazena em um banco de dados. Os dados são então organizados e visualizados em dashboards, proporcionando uma interface gráfica para análise detalhada.

3. Metodologia

O desenvolvimento da solução seguiu uma metodologia estruturada, começando pelo levantamento dos requisitos, que incluiu a identificação das necessidades de monitoramento térmico dentro da cabine de veículos automotivos, especificando a necessidade de medir e visualizar temperatura e umidade em tempo real, além de armazenar esses dados na nuvem para análises posteriores. A partir disso, foram realizados testes em diferentes condições para assegurar a robustez do sistema, como também para demonstrar a implementação do sistema no ambiente real de uso.

3.1. Dispositivos IoT

A camada de dispositivos IoT no projeto inclui um ESP32, um sensor DHT11, e um display LCD. O ESP32 é um microcontrolador utilizado como base para o dispositivo IoT, responsável pela captura de dados gerenciando a comunicação entre o sensor, a interface do usuário e a nuvem. Isso só será possível devido sua integração com rede Wi-Fi que utilizaremos para nos conectarmos ao broker e enviar os dados à rede. O DHT11 é um sensor digital que mede temperatura e umidade, cujas especificações estão inclusas a capacidade de medir temperatura no intervalo de 0°C a 50°C com precisão de ±2°C e umidade relativa do ar no intervalo de 20% e 80% com precisão de ±5%. Enquanto o display LCD exibe os dados coletados em tempo real.

A programação dos dispositivos iniciou com a configuração do ambiente de desenvolvimento, utilizando a IDE Arduino para escrever o código[1] contendo toda estrutura para inicializar e configurar os dispositivos, conectar à rede WiFi, ler os dados do sensor, processar esses dados e enviá-los para o broker MQTT.

A foto abaixo refere-se a prototipação inicial, que envolveu a montagem física dos componentes, incluindo a conexão do sensor DHT11 e do display LCD ao ESP32 em uma protoboard.

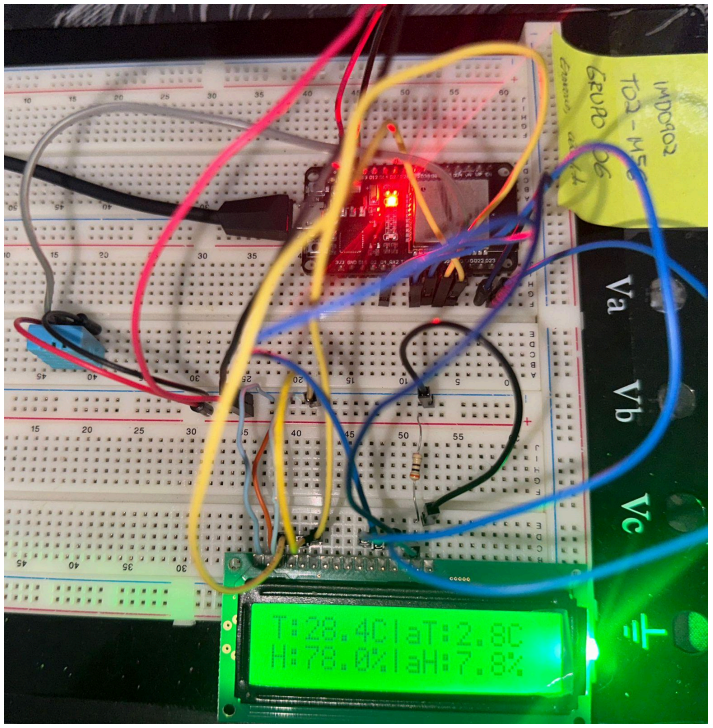


Figura 02: Protótipo da solução inicial. Fonte: Autores.

3.2. Conectividade

No projeto são utilizadas várias tecnologias de comunicação para permitir a coleta, processamento e transmissão de dados entre os dispositivos IoT. As tecnologias de comunicação utilizadas no projeto incluem Wi-Fi para conectividade sem fio, MQTT para a transmissão de dados para o AdaFruit e comunicação serial para a leitura dos dados do sensor e visualização pelo display LCD. Essas tecnologias garantem que o sistema de monitoramento térmico funcione de maneira eficiente e confiável, oferecendo uma solução completa para o monitoramento das condições ambientais dentro da cabine de veículos automotivos.

[1] Github do projeto. Disponível em: <https://github.com/ErnaneJ/itms-iot>

A conectividade Wi-Fi é fundamental para a comunicação sem fio entre o microcontrolador ESP32 e a plataforma de nuvem, permitindo o envio dos dados coletados para armazenamento e análise remota. A configuração da interface envolve a conexão do ESP32 à rede local ou a um ponto de acesso externo, garantindo que o dispositivo esteja online e pronto para enviar e receber dados.

Durante a conectividade Wi-FI existe uma etapa visual de verificação através do LED do ESP32. Ao tentar conectar, o LED pisca a cada 50ms, fica aceso quando conectado e permanece apagado em caso de alguma falha de conexão. Também pode ser observado em vídeo[2].

Além do Wi-Fi, o protocolo MQTT (Message Queuing Telemetry Transport) é utilizado para a comunicação entre o ESP32 e a plataforma de nuvem. O MQTT é um protocolo leve e eficiente projetado para dispositivos IoT, permitindo a publicação e a assinatura de mensagens em tópicos específicos. Isso possibilita o envio dos dados coletados para a nuvem de forma rápida e confiável, garantindo que as informações estejam sempre atualizadas e acessíveis para análise.

O protocolo MQTT é utilizado para enviar os dados de temperatura e umidade para um *broker* MQTT (neste caso, Adafruit IO). A comunicação MQTT é realizada sobre TCP/IP, facilitada pela biblioteca *PubSubClient*. Primeiro, uma conexão TCP é estabelecida com o broker MQTT especificado, neste caso, na plataforma AdaFruit na porta 1883. Uma vez estabelecida a conexão TCP, o protocolo MQTT é utilizado para enviar e receber mensagens. No código[2], a função *connectMQTT()* estabelece e mantém essa conexão, autenticando com credenciais (*mqtt_usernameAdafruitIO* e *mqtt_keyAdafruitIO*) e permitindo que o ESP32 publique dados de temperatura e umidade em tópicos específicos do broker MQTT. Em seguida, no *loop()* os dados são publicados em diferentes tópicos MQTT (por exemplo, Ernane/feeds/actm-temp) se a conexão MQTT estiver estabelecida. Se a conexão falhar, a função *connectMQTT()* é chamada novamente para tentar reconectar.

A comunicação serial também desempenha um papel importante no projeto, permitindo a comunicação entre o ESP32 e o sensor DHT11 para a leitura dos dados de temperatura e umidade. A comunicação serial é usada para estabelecer uma conexão direta entre os dispositivos, garantindo uma transferência de dados rápida e eficiente.

---

[2] Conectividade Wifi ESP32 - IOT. Disponível em: <https://youtube.com/shorts/kTW8ZCR5zig?feature=share>

3.3. Plataformas, Aplicações e Serviços

Para esta aplicação, utilizou-se o protocolo MQTT para enviar dados a um *broker*, especificamente o da Adafruit, conforme discutido anteriormente. Para isso, foi desenvolvida a comunicação no nível de implementação de hardware, considerando um ESP32 com conexão Wi-Fi, utilizando as bibliotecas *wifi.h* e *wificlient.h*. Adicionalmente, a biblioteca *PubSubClient.h* foi empregada para estabelecer a conexão entre a aplicação e o *broker*. A listagem abaixo mostra um breve trecho de código onde é possível realizar uma conexão wi-fi utilizando-se das bibliotecas mencionadas.

```
#include <WiFi.h>
#include <WiFiClient.h>

WiFiClient wifi_client;

// ...

const char* wifi_ssid = "wifi_ssid";
const char* wifi_password = "wifi_password";
int wifi_timeout = 100000;

// ...

void setup(){
  // ...

  WiFi.mode(WIFI_STA); // "Station Mode"
  WiFi.begin(wifi_ssid, wifi_password);

  // ...
}
```

Figura 03 - Listagem - Exemplo de código para conexão *wi-fi* no *ESP32* utilizando-se das bibliotecas *Wifi.h* e *WifiClient.h*. Fonte: Autores.

Agora com o Wi-Fi conectado, basta realizarmos a conexão com o *broker* MQTT e iniciar o processo de publicação dos dados.

```
#include <PubSubClient.h>

// ...
PubSubClient mqtt_client(wifi_client);

const char* mqtt_broker = "io.adafruit.com";
const int mqtt_port = 1883;
int mqtt_timeout = 10000;

const char* mqtt_usernameAdafruitIO = "Ernane";
const char* mqtt_keyAdafruitIO = "<mqtt_keyAdafruitIO>";

void setup(){
  // ...

  mqtt_client.setServer(mqtt_broker, mqtt_port);

  // ...
}
```

Figura 04 - Listagem - Exemplo de código para conexão *MQTT* no *ESP32* utilizando-se da biblioteca *PubSubClient.h*. Fonte: Autores.

Ao final da configuração, chamamos a função denominada *connectMQTT()*. Essa função é responsável por verificar a conexão com o broker e, em caso de desconexão, tentar reconectar. Ela é essencial, pois a cada etapa, precisamos garantir que o broker esteja conectado antes de realizar uma publicação. Isso foi um dos desafios encontrados durante o desenvolvimento.

Como estamos enviando quatro variáveis independentes para feeds diferentes, cada chamada deve verificar se o broker ainda está conectado à aplicação e, se não estiver, aguardar um intervalo de tempo determinado (configurado para 10 segundos) antes de tentar novamente. A publicação é feita por meio de uma simples chamada de função. Um breve exemplo das quatro chamadas, uma para cada *feed*, podem ser visualizadas abaixo:

```
mqtt_client.publish("Ernane/feeds/actm-temp" , String(temp).c_str(), true)
mqtt_client.publish("Ernane/feeds/actm-atemp", String(averageTemp).c_str(), true)
mqtt_client.publish("Ernane/feeds/actm-hum"  , String(hum).c_str(), true)
mqtt_client.publish("Ernane/feeds/actm-ahum" , String(averageHum).c_str(), true)
```

Figura 05 - Listagem - Exemplo de código para envio de dados pelo cliente MQTT. Fonte: Autores.

Note a utilização de *temp*, *averageTemp*, *hum*, e *averageHum*. Todas essas variáveis são populadas e calculadas antes de enviarmos os dados capturados via comunicação serial, utilizando a biblioteca DHT para coleta dos dados do sensor DHT11. A configuração do sensor e a coleta dos dados são extremamente simples com o uso dessa biblioteca. Abaixo está um exemplo de como realizamos essa captura:

```
#define DHTTYPE DHT11
// ...

DHT DHT(dht_pin, DHTTYPE);

float hum = 0.0;
float temp = 0.0;

// ...

void setup(){
  // ...
  pinMode(dht_pin, INPUT);
  DHT.begin();
  // ...
}

void loop(){
  // ...
  temp = DHT.readTemperature();
  hum = DHT.readHumidity();
  // ...
}
```

Figura 06 - Listagem - Exemplo de código para captura de dados do sensor DHT11 utilizando-se da biblioteca *DHT.h*. Fonte: Autores.

Para calcular as médias, ou seja, os dados para *averageTemp* e *averageHum*, é necessário armazenar as informações em um *array* e, a partir dele, realizar o cálculo da média antes de enviar os dados para o broker, conforme descrito anteriormente.

Do lado da aplicação, estamos bem encaminhados. Agora, precisamos configurar o broker da Adafruit. Após acessar o site e criar uma conta, podemos criar os *feeds* necessários simplesmente dando um nome a eles. Os feeds que criamos foram:

Figura 07 - Feed criando para recepção de dados relacionado com a média da umidade capturada pelo sensor. Fonte: Autores/Adafruit. Disponível em: <https://io.adafruit.com/Ernane/feeds/actm-ahum>. Acesso em: 23 de junho de 2024.

Figura 08 - Feed criando para recepção de dados relacionado com a temperatura capturada pelo sensor. Fonte: Autores/Adafruit. Disponível em: <https://io.adafruit.com/Ernane/feeds/actm-temp>. Acesso em: 23 de junho de 2024.

Figura 09 - Feed criando para recepção de dados relacionado com a média da temperatura capturada pelo sensor. Fonte: Autores/Adafruit. Disponível em: <https://io.adafruit.com/Ernane/feeds/actm-atemp>. Acesso em: 23 de junho de 2024.

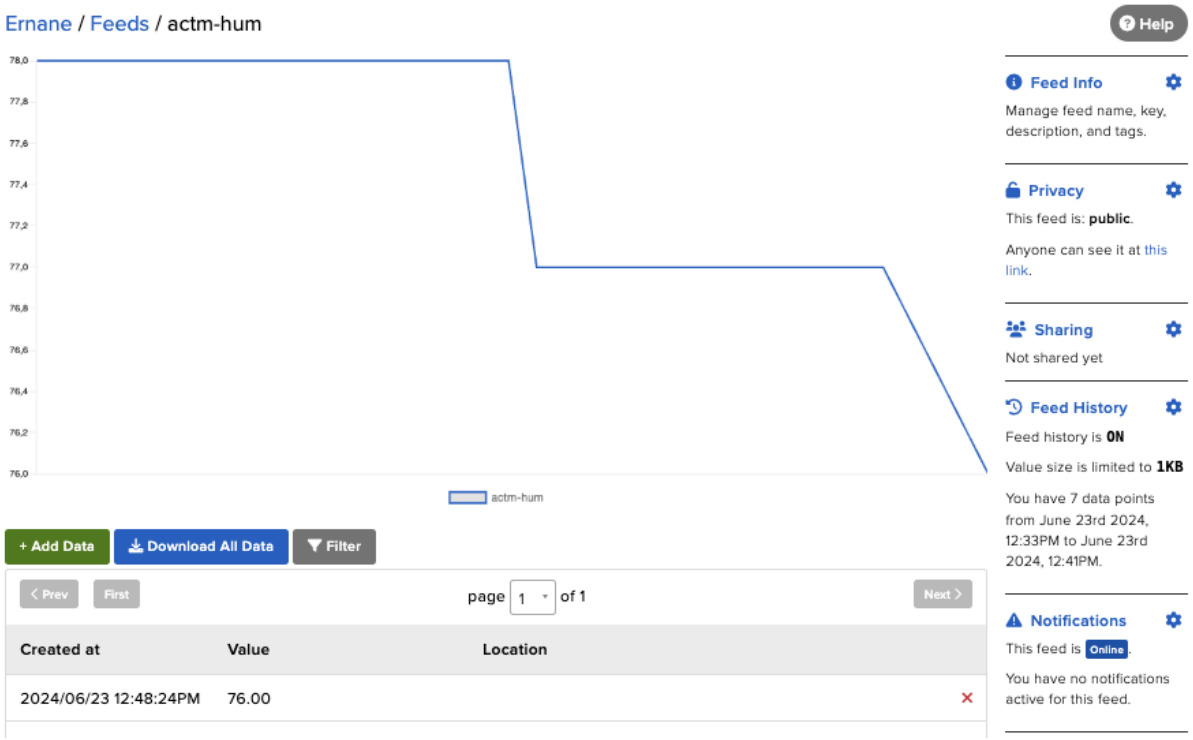


Figura 10 - Feed criando para recepção de dados relacionado com a umidade capturada pelo sensor. Fonte: Autores/Adafruit. Disponível em: <https://io.adafruit.com/Ernane/feeds/actm-hum>. Acesso em: 23 de junho de 2024

Para a conexão, é importante notar que utilizamos tanto um nome de usuário quanto uma chave de acesso. Essa chave pode ser obtida clicando no botão amarelo no topo do site do broker.

Lá, encontramos o nome do nosso usuário, a chave necessária para a conexão, e até mesmo alguns exemplos de como realizar a integração. Essas são as únicas informações que precisamos para nos conectar ao broker e enviar nossos dados além da chave de cada feed que, para o caso base, correspondem aos seus respectivos nomes. No nosso caso, os feeds são:

Feed Name	Key	Last value
actm-ahum	actm-ahum	20.63
actm-atemp	actm-atemp	37.98
actm-hum	actm-hum	27.00
actm-temp	actm-temp	30.90

Figura 11 - Feeds criados para coleta de dados.. Fonte: Autores/Adafruit. Disponível em: <https://io.adafruit.com/Ernane/feeds>. Acesso em: 23 de junho de 2024

Assim, podemos nos conectar ao broker. Com tudo configurado e os dados chegando nos nossos *feeds*, podemos utilizar o servidor da *Adafruit* para apresentá-los de uma forma mais interessante. No painel de cada *feed*, há um gráfico de linha, como mostrado nas imagens acima. No entanto, podemos criar um dashboard interativo para melhorar nossa visualização.

No menu, há a opção "*Dashboard*", onde podemos criar um novo. Nesse *dashboard*, você pode configurar os *widgets* conforme desejar, baseando-se nos dados obtidos e considerando as limitações da plataforma. Para a nossa aplicação, criamos o seguinte *dashboard*:

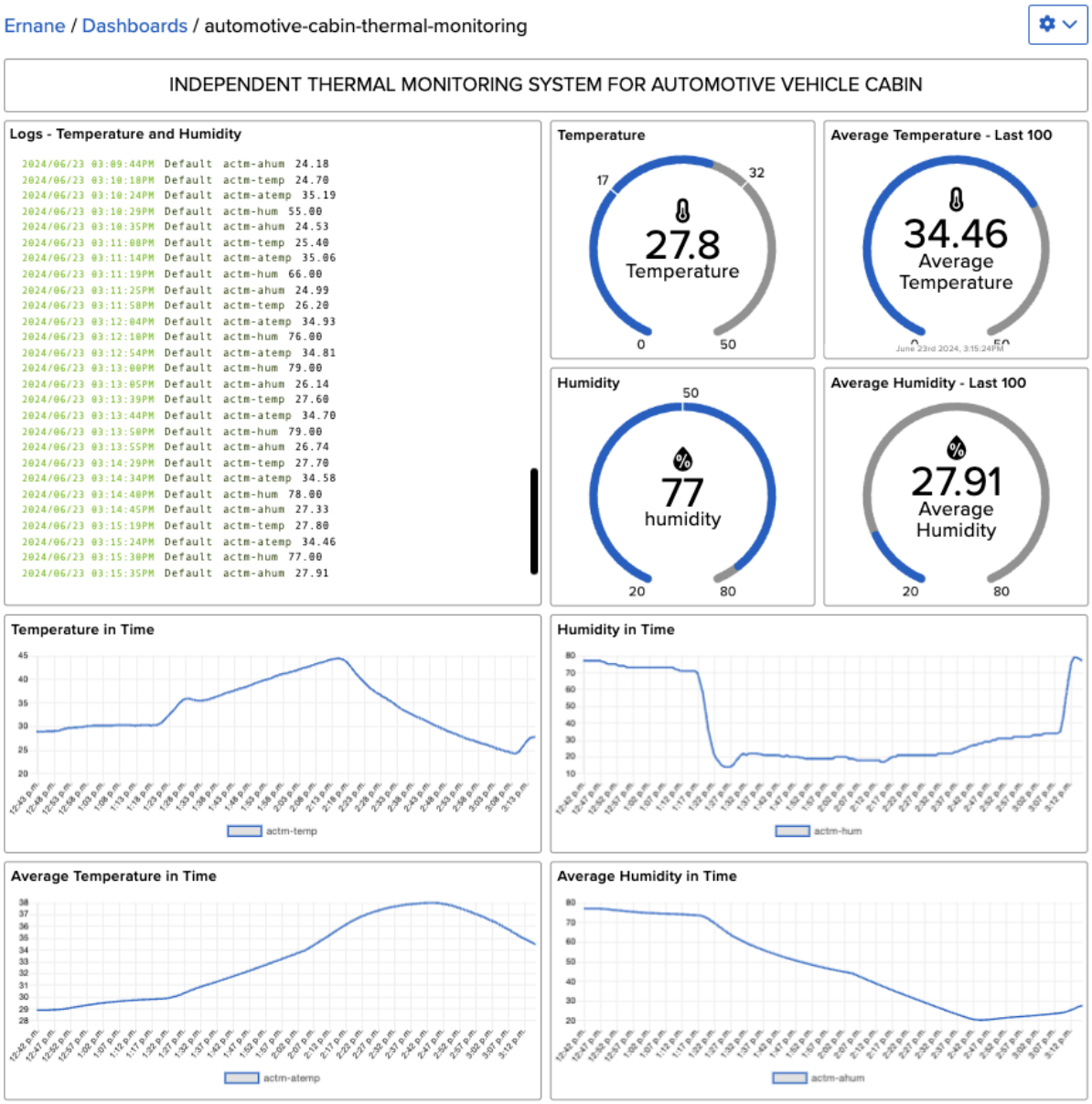


Figura 12 - Dashboard criado para visualização de dados.. Fonte: Autores/Adafruit. Disponível em:

<https://io.adafruit.com/Ernane/dashboards/automotive-cabin-thermal-monitoring>. Acesso em: 23 de junho de 2024

No dashboard, temos *logs* de temperatura e umidade, permitindo visualizar quando uma nova informação foi adicionada pelo *publisher* ao nosso *subscriber*, que, neste caso, é o *broker* da *Adafruit*. Isso é útil para identificar se as informações estão sendo atualizadas consistentemente e se novas requisições foram realizadas. Esses *logs* ajudam a acompanhar em tempo real o que está acontecendo.

Além disso, o dashboard possui quatro mostradores digitais que exibem a temperatura, a umidade e suas respectivas médias. Também incluímos gráficos de linha independentes para cada uma dessas quatro informações, permitindo visualizar a variação ao longo do tempo.

Com esse dashboard, é possível acompanhar todas essas informações em tempo real, garantindo que você possa monitorar o comportamento do dispositivo em relação ao ambiente de qualquer lugar do mundo.

4. Resultados e discussões

Por meio de um processo iterativo de desenvolvimento, teste e refinamento, garantiu-se que a solução fosse robusta e atendesse aos requisitos especificados, proporcionando uma maneira eficaz de monitorar as condições térmicas na cabine de veículos automotivos.

Uma das maiores dificuldades encontradas foi a realização de múltiplas publicações. Ao realizar uma única publicação, notamos que o broker eventualmente poderia desconectar, resultando na perda de informações que deveriam ser enviadas posteriormente. Inicialmente, enviamos apenas a temperatura até compreendermos e resolvermos o problema, permitindo o envio de todas as quatro variáveis.

Outro problema foi a configuração do hardware escolhido, o ESP32, na IDE. Encontramos um problema de compatibilidade com a velocidade de transmissão de informações. Além disso, a taxa de amostragem foi um desafio; se muito rápida, poderíamos ser barrados pelo broker, o que não é desejável. Portanto, estimamos o tempo necessário para enviar uma informação e o intervalo até poder enviar outra, espaçadas em intervalos seguros para evitar "too many requests". No fim, todos os problemas foram resolvidos, permitindo-nos compreender melhor o funcionamento

da conexão e a melhor maneira de compartilhar/publicar as informações necessárias.

Os resultados foram satisfatórios. Conforme comentado no início deste relatório, nosso objetivo era realizar dois testes principais, que foram efetivamente realizados. Considerando a cidade de Parnamirim, Rio Grande do Norte, no dia da coleta de dados, que apresentou tempo quente e nublado, tivemos a oportunidade de realizar os dois testes desejados.

Primeiramente, mantivemos o veículo no sol para capturar a temperatura máxima atingida. Devido à nebulosidade, a temperatura máxima atingida foi de aproximadamente 34°C. Para aprimorar a medição e atingir o limite do sensor, ligamos o aquecedor do veículo no máximo, alcançando 49°C dentro da cabine. O tempo necessário para alcançar 34°C foi entre 12:42 e 13:25. Após ligar o aquecedor, a temperatura subiu para 49°C entre 13:25 e 14:14.

Realizamos um novo teste para capturar o tempo necessário para o veículo atingir uma temperatura confortável de 24°C, o que ocorreu às 15:10. Todo o processo foi realizado de forma remota, permitindo obter a performance do ar do veículo tanto no modo ar-condicionado quanto no modo aquecedor. Além disso, conseguimos obter informações sobre os limites do sensor utilizado, já que, com a temperatura mais alta, a umidade registrada foi de 13%, valor não confiável devido à faixa de operação do sensor ser entre 20% e 80%, segundo seu *datasheet*.

Realizar um experimento mais detalhado, conduzindo vários testes durante diferentes estações do ano para captar os impactos no interior do veículo quando exposto por longos períodos a uma rotina comum e responder, por exemplo, as seguintes perguntas:

1. Qual foi a média de temperatura e umidade ao longo do tempo?

2. O material interno do veículo / carga sensível sofreu algum dano?

Se confirmado que há danos, podemos realizar novos experimentos, considerando o veículo na sombra, usando algum tipo de manta térmica no forro ou simplesmente uma capa protetora que reflita os raios UV. O objetivo é reduzir a temperatura interna e/ou proteger as partes internas e externas do veículo.

Portanto, constatamos que esse dispositivo é útil tanto para avaliar a performance do ar-condicionado do veículo de maneira independente da central do veículo, quanto para realizar uma análise quantitativa da exposição a diferentes

temperaturas, considerando o uso por um usuário comum na cidade de Parnamirim, Rio Grande do Norte.

5. Referências

ADAFRUIT. Adafruit IO. Adafruit, 2024. Disponível em: <https://io.adafruit.com/>. Acesso em: 23 jun. 2024.

MOUSER. Datasheet DHT11. Mouser Electronics, 2024. Disponível em: <https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf>. Acesso em: 23 jun. 2024.

VISHAY. LCD016N002BCFHE/T Datasheet. Vishay Intertechnology, 2024. Disponível em: <https://www.vishay.com/docs/37484/lcd016n002bcfhet.pdf>. Acesso em: 23 jun. 2024.

ESPRESSIF. ESP32 Datasheet. Espressif Systems, 2024. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>. Acesso em: 23 jun. 2024.

ADAFRUIT. ITMS - Average Humidity Feed. Adafruit, 2024. Disponível em: <https://io.adafruit.com/Ernane/feeds/actm-ahum>. Acesso em: 23 jun. 2024.

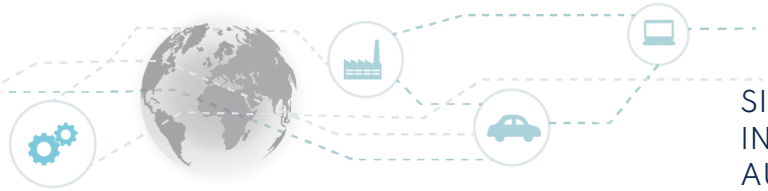
ADAFRUIT. ITMS - Average Temperature Feed. Adafruit, 2024. Disponível em: <https://io.adafruit.com/Ernane/feeds/actm-atemp>. Acesso em: 23 jun. 2024.

ADAFRUIT. ITMS - Dashboard. Adafruit, 2024. Disponível em: <https://io.adafruit.com/Ernane/dashboards/automotive-cabin-thermal-monitoring>. Acesso em: 23 jun. 2024.

ADAFRUIT. ITMS - Humidity Feed. Adafruit, 2024. Disponível em: <https://io.adafruit.com/Ernane/feeds/actm-hum>. Acesso em: 23 jun. 2024.

ADAFRUIT. ITMS - Temperature Feed. Adafruit, 2024. Disponível em: <https://io.adafruit.com/Ernane/feeds/actm-temp>. Acesso em: 23 jun. 2024.

FERREIRA, ROCHA-JUNIOR. Repositório ITMS-IoT. GitHub, 2024. Disponível em: https://github.com/ErnaneJ/itms-iot. Acesso em: 23 jun. 2024.



SISTEMA DE MONITORAMENTO TÉRMICO INDEPENDENTE PARA CABINE DE VEÍCULOS AUTOMOTIVOS