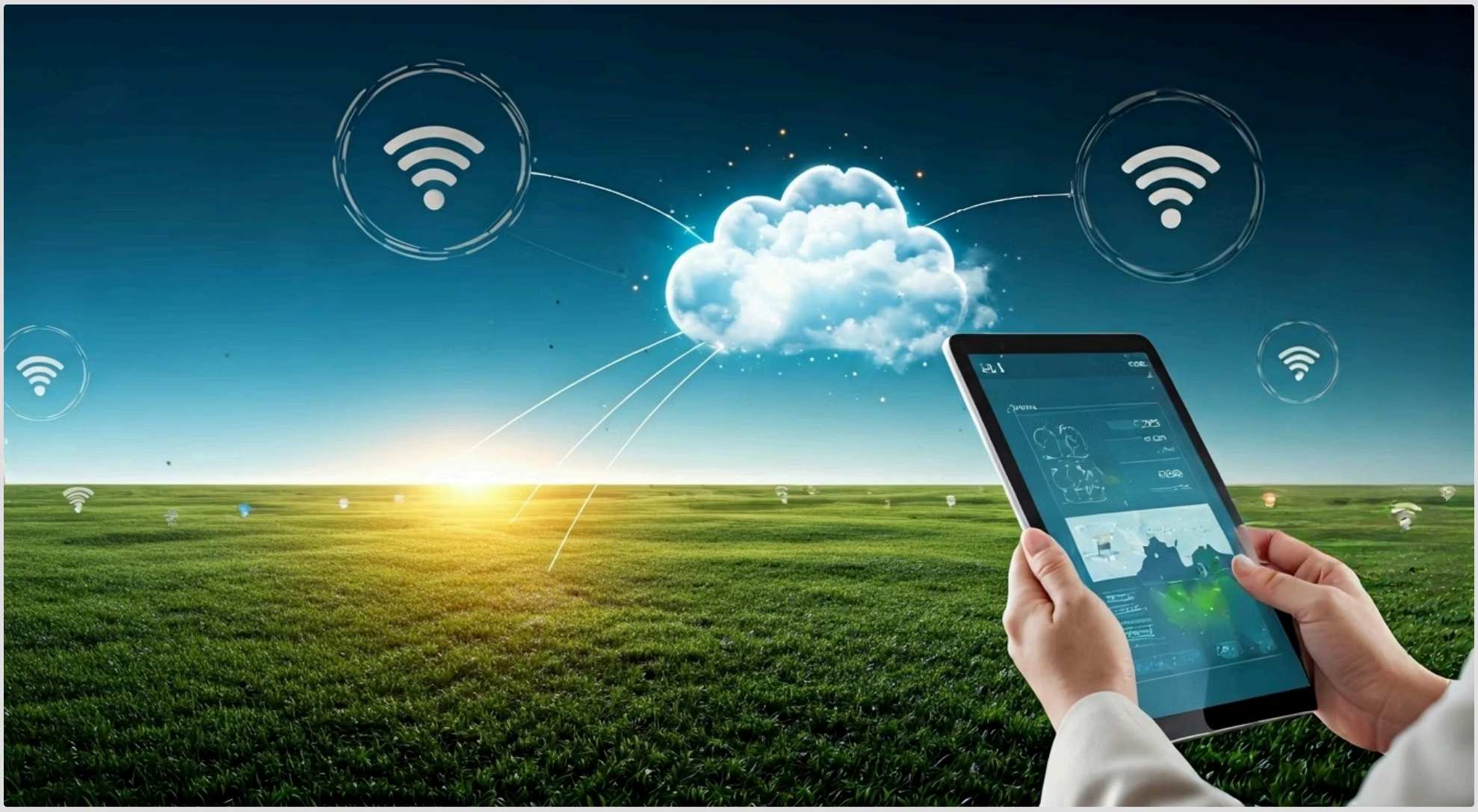
Aula 23 – Projeto 1: Sistema de Monitoramento Ambiental (Parte 1 - Concepção e Hardware)



Bem-vindo à Aula 23 do nosso curso de Desenvolvimento de Aplicações IoT! Hoje, daremos o pontapé inicial em um projeto prático e extremamente relevante: a construção de um sistema de monitoramento ambiental. Em um mundo cada vez mais conectado e consciente da importância dos dados para a tomada de decisões, entender como coletar informações do ambiente ao nosso redor é uma habilidade fundamental. Seja para otimizar o consumo de energia em edifícios, monitorar a qualidade do ar em grandes cidades ou até mesmo para automatizar a irrigação em uma fazenda, a capacidade de "sentir" o ambiente é o primeiro passo.

Esta aula foi cuidadosamente desenhada para você que busca não apenas o conhecimento teórico, mas também a aplicação prática que faz a diferença em seu currículo, seja para horas complementares ou para se destacar em avaliações de títulos. Ao final desta jornada, você será capaz de conceber um projeto IoT do zero, selecionar os componentes de hardware adequados, montar um circuito funcional e desenvolver o firmware inicial para coletar dados de sensores, além de prepará-los para envio. É a base sólida para qualquer aplicação IoT que você venha a desenvolver.

Vamos mergulhar na fase de concepção e hardware, que é onde a mágica começa. Partiremos da ideia de um sistema que mede temperatura, umidade e qualidade do ar, e transformaremos essa ideia em um protótipo funcional. Abordaremos desde a escolha estratégica do microcontrolador e dos sensores até a montagem física e a programação inicial, culminando no envio dos dados para um broker MQTT local. Prepare-se para colocar a mão na massa e ver a teoria se transformar em realidade!

A Gênese do Projeto: Definindo o Problema e os Requisitos

Toda grande inovação começa com uma necessidade, um problema a ser resolvido. No universo da Internet das Coisas (IoT), essa premissa é ainda mais evidente. Antes de sequer pensar em fios, códigos ou placas, precisamos entender o "porquê" do nosso sistema. Por que monitorar o ambiente? Em um cenário global onde a qualidade do ar, as variações climáticas e a eficiência energética são temas urgentes, ter dados precisos e em tempo real é crucial. É aqui que a fase de concepção se torna a espinha dorsal de qualquer projeto bem-sucedido.

Imagine que você está construindo uma casa. Não começaria a colocar tijolos sem um projeto arquitetônico, certo? Da mesma forma, em um projeto IoT, a definição de requisitos é o nosso projeto arquitetônico. Ela nos permite visualizar o que o sistema deve fazer, como deve se comportar e quais são as suas limitações. Essa etapa é fundamental para evitar retrabalho, otimizar recursos e garantir que o produto final atenda às expectativas, seja ele um sistema para uma grande indústria ou um pequeno protótipo para sua casa.

Objetivo Principal

Criar um sistema que possa medir temperatura, umidade e qualidade do ar

Requisitos Funcionais

O que o sistema faz: coleta de dados, processamento local, envio de informações

Requisitos Não Funcionais

Como o sistema faz: autonomia, baixo custo, frequência de atualização, latência

Para o nosso Projeto 1, o objetivo é claro: criar um sistema que possa medir temperatura, umidade e qualidade do ar. Mas isso é apenas o ponto de partida. Precisamos ir além e detalhar o que esperamos. Por exemplo, o sistema deve ser autônomo? Precisa ser de baixo custo? Os dados devem ser atualizados a cada minuto? Essas perguntas nos levam aos requisitos funcionais (o que o sistema faz) e não funcionais (como o sistema faz). Ao detalhar esses pontos, já estamos pensando em como as tendências como **Edge Computing** podem ser aplicadas, processando dados mais perto da fonte para reduzir latência.

Escolha dos Componentes: O Cérebro e os Sentidos do Nosso Sistema

Com os requisitos do nosso sistema de monitoramento ambiental bem definidos, o próximo passo é dar vida a essa ideia, selecionando os componentes de hardware que farão o trabalho pesado. É como montar uma equipe para uma missão: cada membro precisa ter habilidades específicas e complementares para que o objetivo seja alcançado. No nosso caso, precisamos de um "cérebro" para processar as informações e "sentidos" para coletar os dados do ambiente.

ESP32 - O Cérebro

A escolha do microcontrolador é crucial, pois ele será o coração do nosso sistema. Para projetos IoT, o **ESP32** se destaca como uma escolha poderosa e versátil. Pense nele como um pequeno computador de bordo, mas com superpoderes de conectividade. Ele não só possui um processador robusto, capaz de lidar com tarefas complexas, como também integra Wi-Fi e Bluetooth, essenciais para a comunicação sem fio.

- Processador dual-core potente
- Wi-Fi e Bluetooth integrados
- Ideal para Edge Computing
- Baixo consumo de energia

BME680 - Os Sentidos

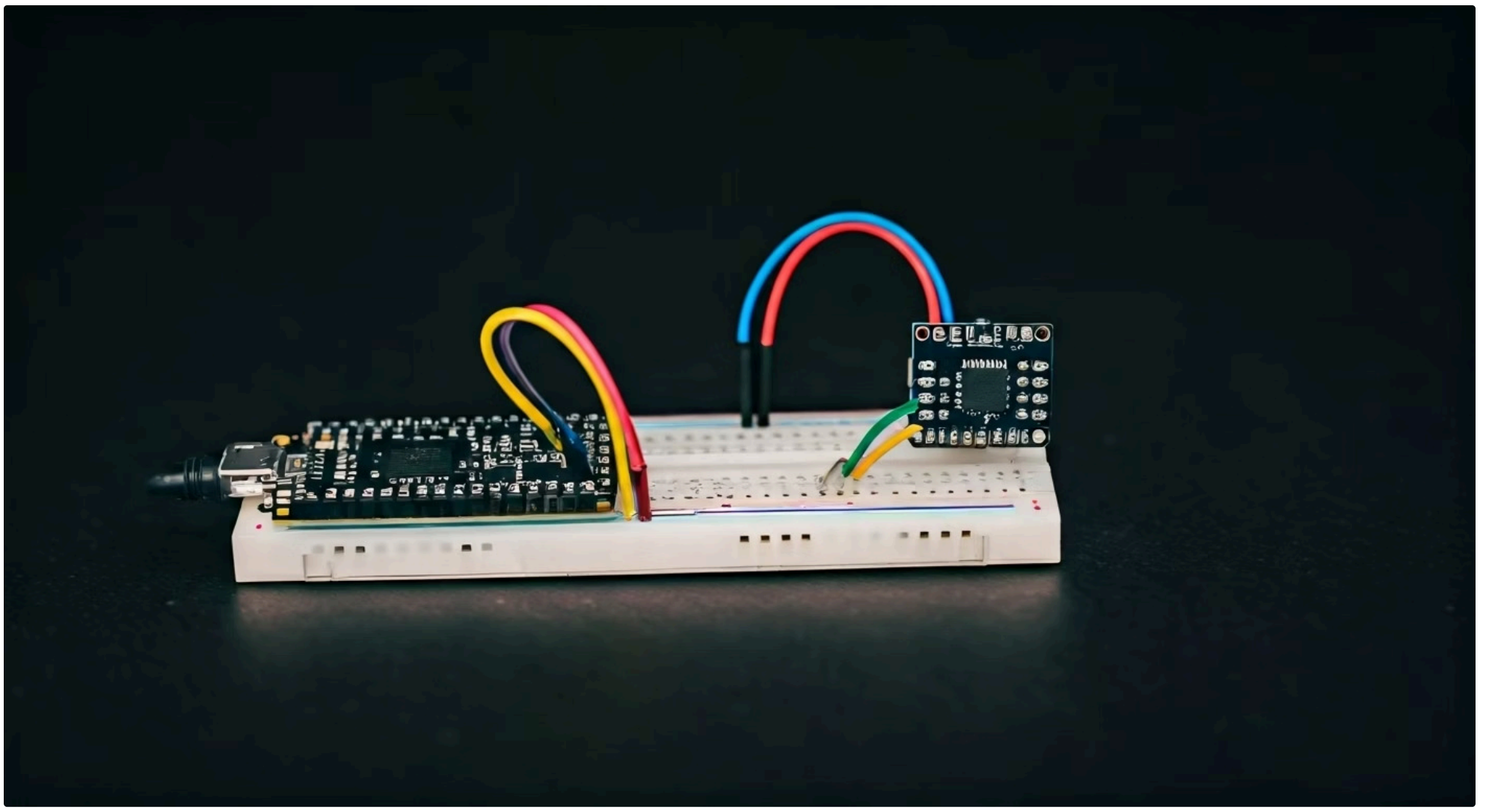
Para os "sentidos" do nosso sistema, que coletarão os dados ambientais, escolhemos o sensor **BME680**. Este é um verdadeiro canivete suíço para monitoramento ambiental, capaz de medir temperatura, umidade, pressão barométrica e, o mais interessante, a qualidade do ar (IAQ - Indoor Air Quality).

- Temperatura ambiente
- Umidade relativa
- Pressão barométrica
- Qualidade do ar (IAQ)

Essa capacidade de processamento na própria "borda" da rede é um pilar do **Edge Computing**, permitindo que o dispositivo tome decisões ou pré-processe dados antes de enviá-los para a nuvem, otimizando o uso de banda e reduzindo a latência. A combinação do ESP32 com o BME680 é um casamento perfeito: o ESP32 oferece a inteligência e a conectividade, enquanto o BME680 fornece dados ambientais ricos e variados. Essa sinergia é a base para explorar conceitos de **AIoT**, onde dados de sensores podem alimentar modelos de Machine Learning para análises mais profundas e tomadas de decisão inteligentes.

Conceito	Âmbito/Aplicação	Base/Origem	Exemplo
ESP32	Microcontrolador para IoT, automação, robótica	SoC (System on Chip) com Wi-Fi e Bluetooth	Controle de dispositivos, coleta de dados, comunicação sem fio
BME680	Sensor ambiental multifuncional	Tecnologia MEMS (Micro-Electro-Mechanical System)	Medição de temperatura, umidade, pressão e qualidade do ar em ambientes

Montando o Circuito: Conectando as Peças do Quebra-Cabeça



Com o cérebro (ESP32) e os sentidos (BME680) em mãos, o próximo passo é fazer com que eles conversem entre si. Pense na montagem do circuito como a construção das vias de comunicação e alimentação que permitirão que esses componentes trabalhem em conjunto. É uma etapa que exige atenção aos detalhes, mas que, uma vez dominada, abre um mundo de possibilidades para a criação de protótipos funcionais. Não se preocupe se você não é um expert em eletrônica; os conceitos básicos são bastante intuitivos.

Protocolo I2C

A conexão entre o ESP32 e o BME680 será feita utilizando o protocolo de comunicação I2C (Inter-Integrated Circuit). Imagine o I2C como uma pequena estrada de mão dupla que permite que vários dispositivos conversem com um único microcontrolador usando apenas dois fios: um para dados (SDA) e outro para o relógio (SCL).

01

Conecte SDA e SCL

Conecte os pinos SDA e SCL do BME680 aos pinos I2C correspondentes no ESP32 (geralmente GPIO 21 para SDA e GPIO 22 para SCL)

02

Conecte a Alimentação

Conecte o VCC do sensor a um pino de 3.3V do ESP32

03

Conecte o Terra

Conecte o GND do sensor ao GND do ESP32

04

Verifique as Conexões

Garanta que todas as conexões estejam firmes e corretas para evitar curtos-circuitos ou falhas na comunicação

Para o nosso projeto, a montagem é relativamente simples. Você conectará os pinos SDA e SCL do BME680 aos pinos I2C correspondentes no ESP32 (geralmente GPIO 21 para SDA e GPIO 22 para SCL, mas sempre verifique a pinagem específica da sua placa ESP32). Em seguida, conecte o VCC do sensor a um pino de 3.3V do ESP32 e o GND do sensor ao GND do ESP32. É crucial garantir que todas as conexões estejam firmes e corretas para evitar curtos-circuitos ou falhas na comunicação. A atenção a esses detalhes também faz parte da **Segurança em IoT**, garantindo a integridade física do dispositivo e a confiabilidade dos dados.

O Primeiro Código: Firmware para Leitura de Sensores

Com o hardware montado e as conexões elétricas estabelecidas, é hora de dar "vida" ao nosso sistema. O firmware é o software que reside no microcontrolador e dita como ele deve interagir com os sensores e o ambiente. É a linguagem que o ESP32 usa para "entender" o que o BME680 está "sentindo". Sem ele, teríamos apenas um conjunto de componentes eletrônicos inertes.

Arduino IDE

Ambiente de desenvolvimento amigável com vasta biblioteca de componentes

Biblioteca Adafruit BME680

Abstrai a comunicação I2C e simplifica a leitura dos dados do sensor

Monitor Serial

Exibe os dados coletados em tempo real para validação

Para desenvolver o firmware, utilizaremos o ambiente de desenvolvimento Arduino IDE (ou PlatformIO, uma alternativa mais robusta). A beleza do ecossistema Arduino é a vasta quantidade de bibliotecas disponíveis, que simplificam enormemente a interação com componentes complexos. Para o BME680, a biblioteca Adafruit BME680 é uma excelente escolha, pois abstrai os detalhes de comunicação I2C e nos permite focar na lógica de leitura dos dados. É como ter um manual de instruções detalhado para cada peça, mas que já vem com as ferramentas certas para usá-las.

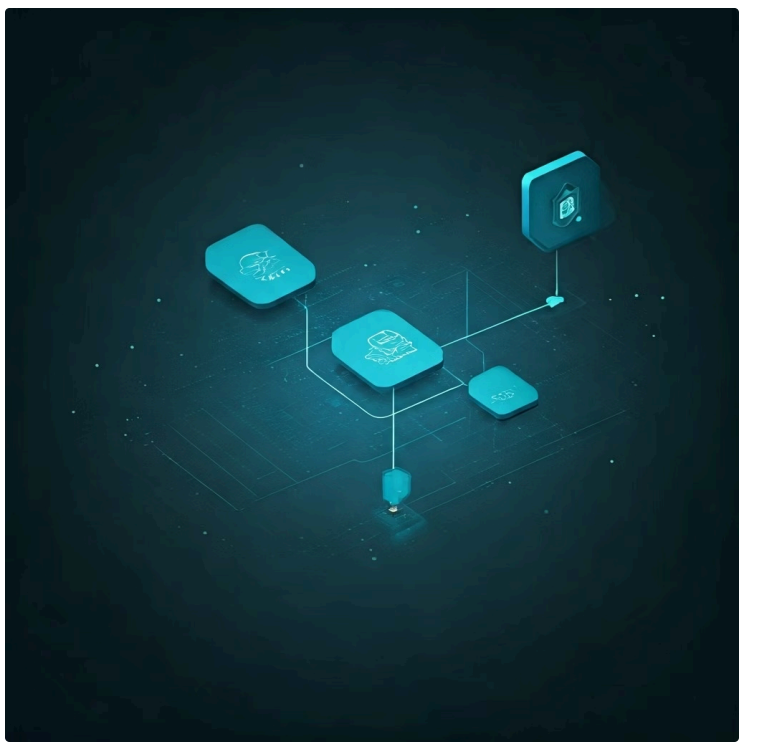
O código inicial terá como objetivo principal configurar o sensor, ler os valores de temperatura, umidade, pressão e qualidade do ar, e exibi-los no monitor serial do computador. Este é o nosso primeiro teste de funcionalidade, a prova de que o hardware e o software estão trabalhando em harmonia. Ao ver os dados fluindo, você terá a certeza de que a base do seu sistema de monitoramento está sólida. Essa leitura inicial no dispositivo é um exemplo prático de **Edge Computing**, onde o processamento primário dos dados ocorre diretamente na fonte, antes de qualquer envio.

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME680.h>

#define SEALEVELPRESSURE_HPA (1013.25)

Adafruit_BME680 bme; // I2C

void setup() {
 Serial.begin(115200);
 while (!Serial);
 Serial.println(F("BME680 Teste de Leitura"));

 if (!bme.begin()) {
 Serial.println(F("Não foi possível encontrar o sensor BME680, verifique as conexões!"));
 while (1);
 }

 // Configurações do sensor (opcional, padrões são bons para começar)
 bme.setTemperatureOversampling(BME680_OS_8X);
 bme.setHumidityOversampling(BME680_OS_2X);
 bme.setPressureOversampling(BME680_OS_4X);
 bme.setIIRFilterSize(BME680_FILTER_SIZE_3);
 bme.setGasHeater(320, 150); // 320 C por 150 ms
}

void loop() {
 if (!bme.performReading()) {
 Serial.println(F("Falha na leitura do sensor!"));
 return;
 }

 Serial.print(F("Temperatura = "));
 Serial.print(bme.temperature);
 Serial.println(F(" *C"));

 Serial.print(F("Umidade = "));
 Serial.print(bme.humidity);
 Serial.println(F(" %"));

 Serial.print(F("Pressão = "));
 Serial.print(bme.pressure / 100.0);
 Serial.println(F(" hPa"));

 Serial.print(F("Gás = "));
 Serial.print(bme.gas_resistance / 1000.0);
 Serial.println(F(" KOhms"));

 Serial.println();
 delay(2000);
}
```

Comunicação em Rede: Introdução ao MQTT

Agora que nosso ESP32 consegue ler os dados do ambiente, o próximo desafio é como fazer com que essas informações cheguem a um lugar onde possam ser armazenadas, visualizadas e analisadas. De que adianta ter um sensor superpreciso se os dados ficam presos dentro do dispositivo? É aqui que entra a comunicação em rede, e no mundo da IoT, um protocolo se destaca pela sua eficiência e leveza: o **MQTT (Message Queuing Telemetry Transport)**.

Imagine o MQTT como um serviço de correio ultraleve, projetado especificamente para enviar pequenas mensagens de forma rápida e confiável, mesmo em redes com largura de banda limitada ou instáveis. Diferente de protocolos mais "pesados" como o HTTP, que exigem uma conexão direta entre cliente e servidor para cada requisição, o MQTT opera com um modelo de "publicação/assinatura" (publish/subscribe). Isso significa que os dispositivos (clientes) não precisam saber quem vai receber suas mensagens; eles simplesmente as publicam em um "tópico". Da mesma forma, outros dispositivos interessados (assinantes) simplesmente "assinam" esses tópicos para receber as mensagens.



Publisher

Dispositivo que publica mensagens em tópicos

Broker

Servidor central que recebe e distribui mensagens

Subscriber

Aplicação que assina tópicos e recebe mensagens

No centro desse sistema está o "broker" MQTT, que atua como o carteiro central. Ele recebe todas as mensagens publicadas e as distribui para todos os assinantes interessados. Essa arquitetura desacoplada é extremamente vantajosa para IoT, pois permite que os dispositivos sejam mais simples, consumam menos energia e sejam mais resilientes a falhas de conexão. É como um jornal: o repórter (publisher) escreve a notícia (mensagem) e a envia para a editora (broker), que a distribui para todos os leitores (subscribers) que assinam aquele jornal (tópico). Essa eficiência é um dos pilares para a escalabilidade de sistemas IoT, especialmente quando pensamos em milhões de dispositivos conectados.

Conceito	Âmbito/Aplicação	Base/Origem	Exemplo
MQTT	Protocolo de mensagens para IoT, telemetria	Modelo publish/subscribe, leve e eficiente	Envio de dados de sensores, controle remoto de dispositivos, notificações
Broker MQTT	Servidor central que gerencia mensagens MQTT	Software que implementa o protocolo MQTT	Mosquitto, HiveMQ, AWS IoT Core, Azure IoT Hub
Tópico MQTT	Canal lógico para categorizar mensagens	String hierárquica (ex: casa/sala/temperatura)	sensores/ambiente/temperatura, atuadores/luz/status

Configurando o Broker MQTT Local: Mosquitto

Para que o nosso sistema de monitoramento ambiental possa enviar seus dados via MQTT, precisamos de um broker – o "carteiro" que receberá e distribuirá as mensagens. Embora existam brokers MQTT na nuvem (como AWS IoT Core ou Azure IoT Hub), para fins de desenvolvimento e testes iniciais, ou mesmo para aplicações de **Edge Computing** onde o processamento local é prioritário, configurar um broker local é uma excelente prática. E entre as opções disponíveis, o **Mosquitto** se destaca pela sua leveza, facilidade de instalação e robustez.

Instalação

Baixe e instale o Mosquitto no seu sistema operacional

Configuração

O broker inicia automaticamente como serviço após instalação

Teste

Use mosquitto_pub e mosquitto_sub para validar funcionamento

Pense no Mosquitto como a central de correios particular do seu projeto. Ele será instalado diretamente no seu computador (ou em um Raspberry Pi, por exemplo) e funcionará como o ponto central para onde o ESP32 enviará os dados e de onde outros aplicativos (como um dashboard futuro) poderão recebê-los. É uma forma de ter controle total sobre o fluxo de mensagens e de testar a comunicação sem depender de uma conexão externa com a internet, o que é ideal para prototipagem e para entender o funcionamento do protocolo.

Comandos de Instalação

Linux (Debian/Ubuntu):

```
sudo apt update && sudo apt install mosquitto mosquitto-clients
```

Windows: Baixe o instalador do site oficial do Mosquitto

A instalação do Mosquitto é bastante direta. Em sistemas baseados em Debian/Ubuntu, um simples comando sudo apt update && sudo apt install mosquitto mosquitto-clients já resolve. Para Windows, basta baixar o instalador do site oficial. Uma vez instalado, o Mosquitto geralmente inicia automaticamente como um serviço. Para testar se ele está funcionando, você pode usar os clientes Mosquitto (mosquitto_pub e mosquitto_sub) para publicar e assinar mensagens no seu próprio broker. Essa configuração local é um passo crucial para construir sistemas IoT resilientes e eficientes, alinhados com as práticas de **Edge Computing**, onde a inteligência é distribuída e não centralizada apenas na nuvem.

Integrando o MQTT ao Firmware: Enviando Dados para o Mundo

Com o Mosquitto configurado e rodando localmente, o palco está montado para o nosso ESP32 começar a "falar" com o mundo exterior. O próximo passo é integrar a funcionalidade MQTT ao firmware que já desenvolvemos para a leitura dos sensores. É como ensinar nosso dispositivo a não apenas ler um livro, mas também a escrever cartas sobre o que aprendeu e enviá-las pelo correio.

Conexão Wi-Fi

O ESP32 se conecta à rede Wi-Fi local

Conexão ao Broker

Estabelece conexão com o broker Mosquitto usando o IP local

Publicação de Dados

Publica os dados dos sensores em tópicos MQTT específicos

Para isso, utilizaremos uma biblioteca MQTT para ESP32, como a PubSubClient. Esta biblioteca simplifica a conexão ao broker, a publicação de mensagens em tópicos e a assinatura de tópicos (caso o ESP32 precise receber comandos). O processo envolve algumas etapas chave: primeiro, o ESP32 precisa se conectar à rede Wi-Fi local. Em seguida, ele estabelece uma conexão com o broker Mosquitto, usando o endereço IP do seu computador (ou do dispositivo onde o Mosquitto está rodando). Uma vez conectado, ele pode começar a publicar os dados dos sensores em tópicos MQTT específicos.

Estrutura de Tópicos

- `sensores/ambiente/temperatura`
- `sensores/ambiente/umidade`
- `sensores/ambiente/pressao`
- `sensores/ambiente/qualidade_ar`

Formato dos Dados

Os dados podem ser enviados como:

- String simples: `"23.5"`
- JSON: `{"temp": 23.5, "unit": "C"}`

Imagine que cada tipo de dado (temperatura, umidade, qualidade do ar) terá seu próprio "canal de notícias" (tópico). Por exemplo, sensores/ambiente/temperatura para a temperatura, sensores/ambiente/umidade para a umidade, e assim por diante. No loop principal do firmware, após ler os dados do BME680, o ESP32 formatará esses valores em uma mensagem (geralmente uma string ou JSON) e os publicará nos tópicos correspondentes. Essa é a essência da comunicação IoT: dados brutos se transformam em informações úteis que podem ser consumidas por outros sistemas. Essa capacidade de enviar dados de forma estruturada é o que permite a integração com plataformas de **AIoT** no futuro, onde algoritmos de inteligência artificial podem analisar esses fluxos de dados para identificar padrões ou anomalias.

```
#include <WiFi.h>
#include <PubSubClient.h>
// ... (incluir bibliotecas do BME680 e Wire)

const char* ssid = "SEU_WIFI_SSID";
const char* password = "SUA_SENHA_WIFI";
const char* mqtt_server = "IP_DO_SEU_BROKER_MOSQUITTO"; // Ex: "192.168.1.100"

WiFiClient espClient;
PubSubClient client(espClient);

// ... (código de setup e loop do BME680)

void setup() {
 // ... (inicialização serial e BME680)

 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
 delay(500);
 Serial.print(".");
 }
 Serial.println("\nWiFi conectado!");
 Serial.print("Endereço IP: ");
 Serial.println(WiFi.localIP());

 client.setServer(mqtt_server, 1883); // Porta padrão MQTT
 // client.setCallback(callback); // Para receber mensagens, se necessário
}

void loop() {
 if (!client.connected()) {
 reconnect(); // Função para reconectar ao MQTT
 }
 client.loop(); // Mantém a conexão MQTT ativa

 if (!bme.performReading()) {
 Serial.println(F("Falha na leitura do sensor!"));
 return;
 }

 // Publica temperatura
 String tempPayload = String(bme.temperature);
 client.publish("sensores/ambiente/temperatura", tempPayload.c_str());
 Serial.print("Publicado temperatura: ");
 Serial.println(tempPayload);

 // Publica umidade
 String humPayload = String(bme.humidity);
 client.publish("sensores/ambiente/umidade", humPayload.c_str());
 Serial.print("Publicado umidade: ");
 Serial.println(humPayload);

 // ... (publicar outros dados como pressão e gás)

 delay(5000); // Publica a cada 5 segundos
}

void reconnect() {
 while (!client.connected()) {
 Serial.print("Tentando conectar ao MQTT...");
 if (client.connect("ESP32Client")) { // ID do cliente
 Serial.println("conectado!");
 // Se houver tópicos para assinar, faça aqui
 // client.subscribe("comandos/luz");
 } else {
 Serial.print("falhou, rc=");
 Serial.print(client.state());
 Serial.println(" tentando novamente em 5 segundos");
 delay(5000);
 }
 }
}
```

Consolidação e Próximos Passos

Chegamos ao final da primeira parte do nosso Projeto 1: Sistema de Monitoramento Ambiental. Nesta aula, você percorreu um caminho essencial, desde a concepção de uma ideia até a sua materialização em hardware e software. Começamos definindo os requisitos do nosso sistema, o que nos permitiu escolher os componentes ideais: o poderoso ESP32 como cérebro e o versátil BME680 como os sentidos. Em seguida, montamos o circuito, estabelecendo as conexões físicas para que esses componentes pudessem interagir. Demos vida ao hardware com o firmware inicial, programando o ESP32 para ler os dados do BME680. Finalmente, introduzimos o protocolo MQTT e configuramos um broker local (Mosquitto), integrando-o ao nosso firmware para que os dados pudessem ser enviados para fora do dispositivo.

Concepção

Definição clara de requisitos funcionais e não funcionais do sistema

Hardware

Seleção estratégica de componentes (ESP32 + BME680) e montagem do circuito

Firmware

Desenvolvimento do código para leitura de sensores e processamento local

Comunicação

Implementação do protocolo MQTT para envio de dados ao broker

Em prática

Você agora tem a base para criar qualquer sistema IoT de coleta de dados. A capacidade de conceber, selecionar componentes, montar e programar a camada de hardware e comunicação é uma habilidade valiosa. Lembre-se que a segurança, tanto física quanto lógica, deve ser uma preocupação constante desde a concepção. A familiaridade com Edge Computing e a preparação para AIoT são diferenciais que você já começou a construir.

Autoavaliação

1 **Qual a principal vantagem do protocolo MQTT em comparação com HTTP para aplicações de Internet das Coisas (IoT)?**

1. Maior complexidade de implementação e uso de banda.
2. Modelo de requisição/resposta síncrono, ideal para dados em tempo real.
3. Leveza, modelo publish/subscribe e eficiência para redes com largura de banda limitada.
4. Exige conexões persistentes e alto consumo de energia dos dispositivos.

2 **No contexto do Projeto 1, qual a função primordial do ESP32?**

1. Apenas atuar como fonte de energia para os sensores.
2. Ser o "cérebro" do sistema, processando dados e gerenciando a comunicação.
3. Exclusivamente medir temperatura e umidade do ambiente.
4. Servir como um broker MQTT embutido para a nuvem.

3 **A fase de "Definição de Requisitos" em um projeto IoT é comparada à construção de uma casa sem um projeto arquitetônico. Qual a principal implicação dessa analogia?**

1. A casa será construída mais rapidamente, pois não há burocracia.
2. A falta de um projeto detalhado pode levar a retrabalho e não atender às necessidades.
3. O custo da construção será significativamente menor sem um planejamento prévio.
4. A casa terá um design mais inovador e flexível.

4 **Qual das seguintes tendências foi incorporada na discussão sobre o processamento de dados mais perto de onde são gerados, visando reduzir latência e consumo de banda?**

1. Blockchain em IoT.
2. Realidade Aumentada (RA).
3. Edge Computing.
4. Computação Quântica.

> **Questão Dissertativa:** Descreva a importância da escolha do sensor BME680 para o Projeto 1 e como suas capacidades multifuncionais contribuem para a riqueza dos dados coletados.

Gabarito e Recursos Adicionais

Gabarito

1

Resposta: c) Leveza, modelo publish/subscribe e eficiência para redes com largura de banda limitada.

2

Resposta: b) Ser o "cérebro" do sistema, processando dados e gerenciando a comunicação.

3

Resposta: b) A falta de um projeto detalhado pode levar a retrabalho e não atender às necessidades.

4

Resposta: c) Edge Computing.

Recursos Adicionais

- **Documentação oficial do ESP32:** Para aprofundar-se nas especificações técnicas do microcontrolador.
- **Datasheet do BME680:** Para entender todos os detalhes e capacidades do sensor.
- **Site oficial do Mosquitto:** Para informações sobre instalação e configuração avançada do broker.
- **Tutorial sobre MQTT:** Para explorar mais a fundo o protocolo e suas aplicações.

Próxima Aula

Aula 24 – Projeto 1: Sistema de Monitoramento Ambiental (Parte 2 - Nuvem e Dashboard)

Levaremos os dados do nosso broker local para a nuvem e construiremos um dashboard para visualizá-los de forma intuitiva.

NOTA IMPORTANTE: As informações regulatórias/legais/técnicas desta aula estão atualizadas até 2025. Consulte sempre fontes oficiais para verificar alterações.