Projeto Final da Fase 2: Etapa 4 – Projeto Final

Aluno: Antonio Carlos Ferreira de Almeida

Data: 19/09/2025

Sumário

3 - https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf..... 20

4- https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf..... 20

5 - https://datasheets.raspberrypi.com/picow/connecting-to-the-internet-with-pico-w.pdf 20

6 - **Network Programmability and Automation, 2nd Edition.** By **Matt Oswalt**, **Christian Adell**, **Scott S. Lowe**, **Jason Edelman**..... 20

7 - **TCP/IP Illustrated, Volume 1: The Protocols, 2nd Edition** By **Kevin R. Fall**, **W. Richard Stevens** 20

8- **How Linux Works, 3rd Edition** By **Brian Ward** 20

9 - **The Linux Command Line, 2nd Edition** By **William E. Shotts** 20

10 - **NGINX HTTP Server - Fifth Edition** 20

Índice de figuras

Índice de Tabelas

1 – Apresentação do projeto.

Arquitetura IoT com BitDogLab, Servidor Ubuntu e AWS/CloudFlare para Automação Segura, utilizando Nuvem, bem como SSL/TLS.

Este projeto propõe uma arquitetura IoT voltada para **AUTOMAÇÃO E CONTROLE de sensores, bem como,** sua utilização segura em qualquer lugar do planeta que possuir uma conexão com **a nuvem**, indo desde a possibilidade de ligar e desligar cargas a análises avançadas e geração de inteligência com dados coletados em tempo real.

A solução integra em 3 etapas:

- **BitDogLab (Raspberry Pi Pico W**) rodando servidor baseado em lwIP, simplificado para microcontroladores. Através de uma abordagem radical será construída toda a lógica de funcionamento do projeto, relativo a esta etapa, sobre o esqueleto de uma aplicação já existente no SDK a saber: “pico_w/wifi/tcp_server/picow_tcp_server.c”,
    - **Ele cria um servidor TCP que:**
        - **Abre uma porta (4242)** *e fica aguardando conexões de clientes.*
        - **Quando um cliente conecta,** *o servidor gera um buffer aleatório de 2048 bytes e envia para o cliente.*
        - **O cliente** *deve devolver esse mesmo buffer de volta.*
        - **O servidor** *compara byte a byte para garantir que o envio e o recebimento funcionam corretamente.*
        - **Repete** *o processo por 10 iterações (TEST_ITERATIONS).*
        - **Se tudo der certo,** *imprime "test success", senão marca como falha.*
    - O esqueleto em si não oferece nenhuma funcionalidade prática. Ele serve apenas como ponto de partida, sobre o qual toda a implementação dos requisitos do projeto será construída. Isoladamente, não traz vantagens além das descritas anteriormente.
      Como exemplo, o código atual apresenta uma versão stand-alone, enquanto o projeto final terá como objetivo demonstrar conectividade em um contexto global.
- **Servidor Ubuntu headless** + Nginx na mesma rede local, atuando como gateway, proxy reverso e ponto de consolidação;
- **Nuvem (https://)** garantindo que a comunicação seja segura, confiável e protegida contra ataques “man-in-the-middle" (MITM), DDoS, injeção SQL, entre outros.

Título do projeto: - DataHarborIoT.

Inspiração:
A ideia veio da analogia com sistemas físicos de armazenamento e transporte (como portos marítimos) e a necessidade de um “ponto de coleta local” para dados IoT, antes

deles seguirem para a nuvem — um lugar confiável e estruturado para “ancorar” e organizar esses dados.

Além disso, “Harbor” também remete a segurança, proteção e estabilidade — tudo que queremos para o armazenamento e gerenciamento de dados sensíveis em IoT.

2 – Motivação.

O **Raspberry Pi Pico W** usado pela BitDogLab é um dispositivo de baixo custo e consumo, capaz de interagir com sensores e atuadores de forma eficiente, mas não projetado para exposição direta à internet devido a restrições de segurança e processamento.

Um **servidor Ubuntu** — especialmente em hardware robusto como **Intel Xeon 28 núcleos e 64 GB de RAM** — oferece recursos para:

- Gerenciamento seguro de conexões;
- Aplicação de firewall e autenticação;
- Roteamento inteligente e pré-processamento de dados;
- Integração direta com APIs e serviços de nuvem para armazenamento massivo.

Obs.: o projeto foi testado com substituição ao ***Intel Xeon,*** por um computador com menos recursos de hardware: Raspberry Pi 5 4Gb com unidade de armazenamento SATA SCSI, SE MOSTRANDO TOTALMENTE PORTÁVEL E USUAL independente da mudança de arquitetura x86 para ARM, rodando Ubuntu Server 24.04.3 LTS (64-bit).

O **Tunnel** atua como elo seguro entre o servidor local e o provedor de nuvem, assim, não há exposição de IPs reais e ainda, agregando funcionalidades como SSL/TLS, mitigação de ataques e otimização de tráfego.

3 – Arquitetura Proposta

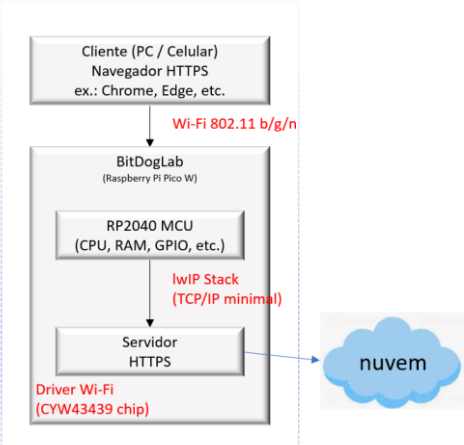

Figura 1: Arquitetura.

1. BitDogLab (Raspberry Pi Pico W)

- Expõe IP, bem como Hostname de forma a se tornar plug-and-play a outros ocupantes da rede, captura dados de múltiplos sensores e permite comutação de cargas através de seus GPIO´s;
- Servidor HTTP embutido para envio dos dados ao Servidor Ubuntu via rede local;
- Processamento mínimo, priorizando baixa latência de envio.

O lwIP é uma pequena implementação independente do conjunto de protocolos TCP/IP.

O foco da implementação TCP/IP do lwIP é reduzir o uso de recursos, mantendo um TCP em escala completa. Isso o torna adequado para uso em sistemas embarcados com dezenas de quilobytes de RAM livre e espaço para cerca de 40 Kb de ROM de código.

Os principais recursos incluem:

  - Protocolos: IP, IPv6, ICMP, ND, MLD, UDP, TCP, IGMP, ARP, PPPoS, PPPoE
  - Cliente DHCP, cliente DNS (incl. resolvedor de nome de host mDNS), AutoIP/APIPA (Zeroconf), agente SNMP (v1, v2c, v3, suporte a MIB privado e compilador MIB)
  - APIs: APIs especializadas para desempenho aprimorado, API de soquete Berkeley-alike opcional
  - Recursos estendidos: encaminhamento de IP por várias interfaces de rede, controle de congestionamento TCP, estimativa de RTT e recuperação/retransmissão rápidas
  - Aplicativos adicionais: servidor HTTP(S), cliente SNTP, cliente SMTP(S), ping, servidor de nomes NetBIOS, respondedor mDNS, cliente MQTT, servidor TFTP

O lwIP é licenciado sob uma licença estilo BSD: http://lwip.wikia.com/wiki/License .

O código Contrib foi movido para o repositório principal, subdiretório 'contrib'.

Construção do Github CI lwip master: https://github.com/lwip-tcpip/lwip/actions

Data de registro: qui 17 out 2002 21:13:13

Licença: Licença BSD Modificada.

Estado do desenvolvimento: 5 - Produção/Estável

2. Servidor Ubuntu Headless (Intel Xeon)

- Atua como **gateway** e **proxy reverso** + Nginx;
- Com capacidade de receber dados de múltiplos Picos, consolidar, validar pacotes e retransmitir pela rede interna ao Docker;
- Encaminha dados para a nuvem via APIs seguras;
- Garante resiliência e alta disponibilidade;
- Pode executar *batch processing* inicial ou compactação antes do envio.

3. Tunnel para receber URL pré-assinada
ex.: (https://meu-bucket.s3.amazonaws.com/dados.json - necessário ter uma conta AWS, bem como iniciar uma VM, EC2 com requisitos de armazenagem e clock**)** o mesmo se faz ao optar por **CloudFlare** *(ambas têm o mesmo papel, más deve-se escolher uma em detrimento a outra, acarretando prós e contras para os dois lados)***:**

- Protege a entrada de dados na nuvem;
- Evita exposição direta do servidor e IP;
- Garante criptografia ponta-a-ponta;
- Facilita escalabilidade global.

4 – Objetivo Estratégico.

Trata de servir páginas a usuários finais e construir um **pipeline robusto, seguro e criptografado de ponta-a-ponta, como sugere as melhores práticas para construção de soluções IoT**.
O foco é criar uma **fonte única de verdade** (*single source of truth*) para os dados captados, permitindo que áreas especializadas possam:

1. Filtrar apenas os dados relevantes à sua atuação;
2. Realizar análises históricas e preditivas;
3. Alimentar modelos de machine learning e sistemas de decisão.
4. Controlar cargas a partir de qualquer lugar do planeta com conectividade.

b) Especificação do hardware.

1 – Diagrama em blocos.

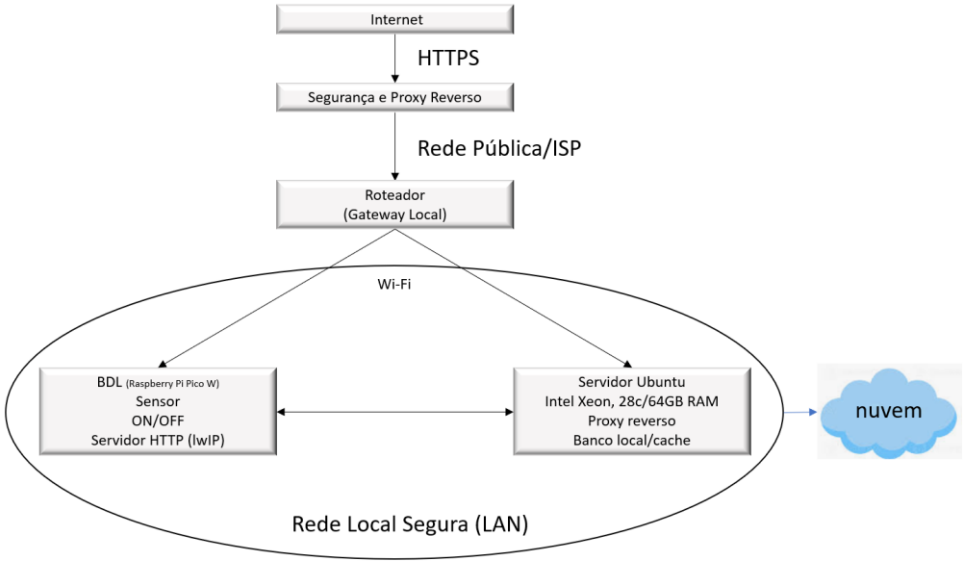

Figura 2: Diagrama em Blocos.

2 – Função de cada bloco.

1. BDL – inicialização, gateway, sensoriamento;

2. Rede mundial é a via de mão dupla onde trafegam os dados;

3. Servidor Ubuntu (meio entre coleta, segurança e nuvem);

4. Wi-Fi/Roteador mantém a LAN com trafego entre BDL e o Servidor;

5. Nuvem torna tudo possível roteando produtor e consumidor fornecendo páginas para interação e dados.

3 – Configuração de cada bloco.

Sistema de Controle (Raspberry Pi Pico W) Bitdoglab

1. Inicialização do Dispositivo Raspberry Pi Pico W (BDL)

   - ✓ Iniciar servidor HTTP simplificado (lwIP).
   - ✓ Inicializar sensores e interfaces de captura.
   - ✓ Definir metadados fixos (ID dispositivo, localização, etc).
   - ✓ Aguardar evento de coleta periódica ou acionamento externo.

2. Captura de Dados no Pico

   - ✓ Ler sensores físicos (grandezas físicas: temperatura, umidade, etc).
   - ✓ Montar pacote de dados com:
     - Valores dos sensores;
     - Timestamp atual;
     - Metadados de origem.
   - ✓ Armazenar pacote temporariamente para envio.

3. Envio dos Dados para o Servidor Ubuntu (via HTTP local)

   - ✓ Pico realiza requisição HTTP POST para servidor na rede local.
   - ✓ Dados são enviados no formato JSON.
   - ✓ Esperar resposta de confirmação do recebimento.

4. Recepção e Validação dos Dados no Ubuntu

   - ✓ Receber requisição HTTP.
   - ✓ Validar integridade e formato do pacote.
   - ✓ Aplicar autenticação e regras de segurança.
   - ✓ Armazenar dados temporariamente localmente (cache local ou Postgres).

5. Pré-processamento e Consolidação (apenas onde há leitura de sensor)

   - ✓ Agregar dados recebidos de múltiplos dispositivos (funcionalidade possível, más não implementada na prova de conceito)
   - ✓ Executar compressão ou filtragem simples.
   - ✓ Preparar pacote para envio seguro à nuvem.

6. Comunicação Segura via Tunnel (https://)

   - ✓ Estabelecer conexão segura e criptografada com a nuvem.
   - ✓ Transmitir dados para o serviço via APIs protegidas.
   - ✓ Confirmar recebimento e sucesso da operação.
   - ✓ Retornar ao ciclo inicial...

4 – Formato do Pacote (8 bytes).

Byte	Campo	Descrição
0	**STX**	Início do pacote (0x7E)
1	**ID**	Código do sensor/dispositivo (0–255)
2	**Tipo**	Tipo de dado (0=temp, 1=pressão, 2=umidade, 3=ligado/desligado)
3-4	**Valor**	Dado em **inteiro de 16 bits** (escala fixa para todos)
5	**Flags**	Bits extras (ex.: ligado/desligado, unidade de medida)
6	**CRC**	Soma dos bytes 1–5 (mod 256)
7	**ETX**	Fim do pacote (0x7F)

Figura 3: Estrutura de Pacote.

- **Convenção dos Dados:**
  - **Temperatura**: Valor × 100 (ex.: 25,37°C → 2537)
  - **Pressão**: Valor × 10 (ex.: 1013,2 hPa → 10132) (possível, más não implementado)
  - **Umidade**: Valor × 100 (ex.: 65,4% → 6540) (possível, más não implementado)
  - **Ligado/Desligado**: Valor 1 = ligado, 0 = desligado
- **Tamanho:**
  - Todos usam **int16_t**, assim o parsing é o mesmo.
- **Exemplo de Pacote:**
  - [0] 0x7E (STX)
  - [1] 0x02 (ID = 2)
  - [2] 0x00 (Tipo = temperatura)
  - [3] 0x09 (Valor alto = 2537 >> 8)
  - [4] 0xE9 (Valor baixo = 2537 & 0xFF)
  - [5] 0x00 (Flags = não usado)
  - [6] 0xF4 (CRC = (0x02+0x00+0x09+0xE9+0x00) % 256)
  - [7] 0x7F (ETX)
- **Estrutura:**

```c
typedef struct {
    uint8_t start;
    uint8_t id;
    uint8_t type;
    int16_t value;
    uint8_t flags;
    uint8_t crc;
    uint8_t end;
} DataPacket;
```

- **Cálculo CRC:**

```c
uint8_t calc_crc(uint8_t id, uint8_t type, int16_t value, uint8_t flags) {
    uint8_t sum = id + type + ((value >> 8) & 0xFF) + (value & 0xFF) + flags;
    return sum % 256;
}
```

- **Construtor:**

```c
DataPacket create_packet(uint8_t id, uint8_t type, int16_t value, uint8_t flags) {
    DataPacket p;
    p.start = STX;
    p.id = id;
    p.type = type;
    p.value = value;
    p.flags = flags;
    p.crc = calc_crc(p.id, p.type, p.value, p.flags);
    p.end = ETX;
    return p;
}
```

- Apoio:

Parsing no Receptor:

O receptor lê byte a byte:

1. Espera STX;
2. Lê os próximos 6 bytes;
3. Confere ETX;
4. Calcula CRC e valida;
5. Converte value de volta para a grandeza original.

```c
void send_packet(DataPacket *p) {
    uint8_t *ptr = (uint8_t *)p;
    for (int i = 0; i < sizeof(DataPacket); i++) {
        putchar(ptr[i]); // Pode ser substituído por UART, SPI, etc.
    }
}
```

5 – Lista de materiais (uso de sensores limitado para demonstração)**.**

Quantidade	*Descrição*
01	**BitDogLab** (**Raspberry Pi Pico W)**
01	**Cabo micro-USB** (para alimentação e programação)
01	**Temperatura** – Sensor interno
01	**Fonte de alimentação 5V USB**
...	**Ferramentas e Acessórios**

Tabela 1: Lista de Materiais.

7 – Circuito completo do hardware.

- Acionamento liga/desliga com foco no GPIO 25, led da placa apenas servindo de demonstração, contudo, qualquer GPIO livre pode ser ligado a uma carga para controle.
- Para ilustrar a capacidade de coleta de dados, utilizamos o adc ligado ao sensor de temperatura do chip, contudo, trata-se de demonstrar que qualquer sensor, respeitadas as características do Pico W, podem ser suplentes.

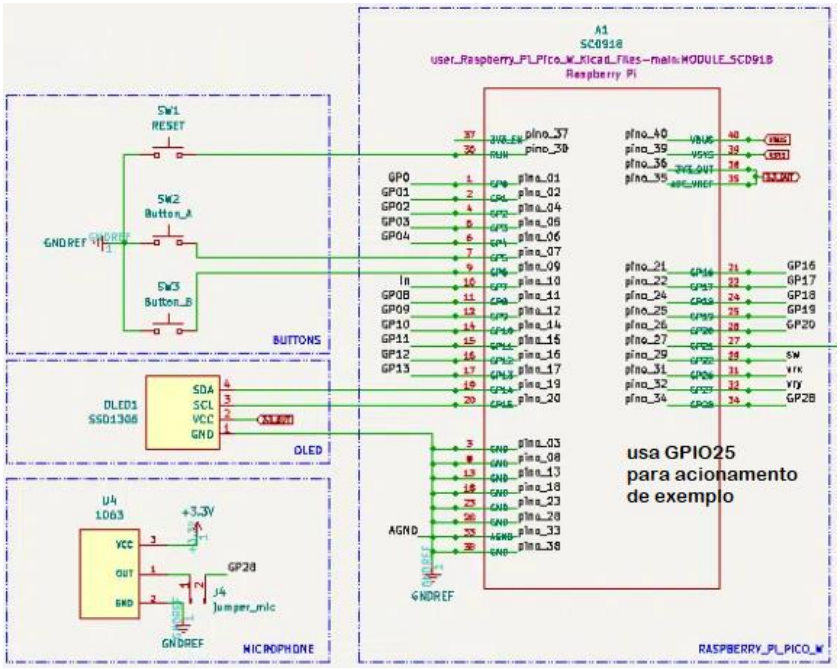

Figura 4: Circuito completo.

c) Especificação do firmware.

1 – Blocos funcionais.

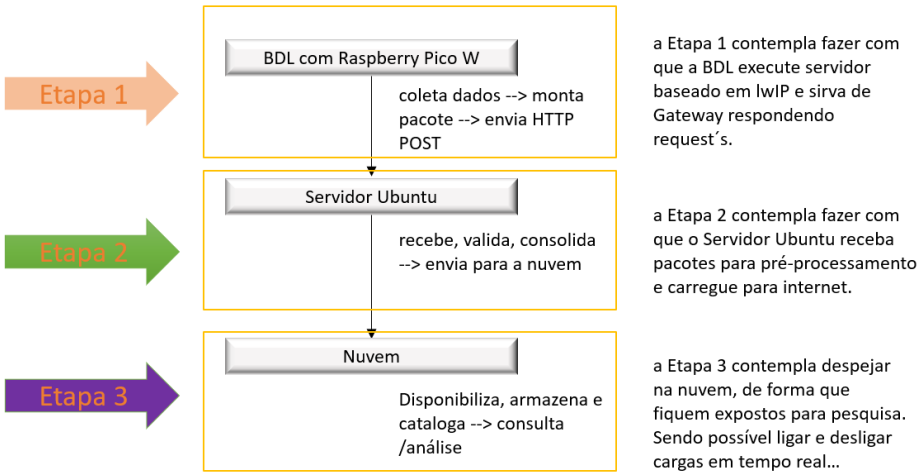

Figura 5: Blocos Funcionais.

1. BitDogLab (BDL) com Raspberry Pi Pico W

Função:

- Captura dados sensoriais (grandezas físicas como: temperatura, umidade, etc).
- Anexa metadados: timestamp, ID do dispositivo, localização, contexto da coleta.
- Executa servidor HTTP leve usando a stack lwIP para comunicação local.
- Envia pacotes estruturados para o gateway via rede local.
- Prioriza baixo consumo e latência, sem processamento pesado.
- Capacidade de ligar e desligar cargas.
- Dados podem ser consultados, cargas podem ser comutadas por meio do navegador de sua escolha (testado com Google Chrome e FIREFOX).

Interfaces:

- A título de demonstração usa sensor interno ao chip, contudo, pequenas adaptações ao código possibilitam Sensores físicos (GPIO, I2C, SPI).

2. Gateway Local: Servidor Ubuntu Headless (Intel Xeon 28 núcleos, 64 GB RAM)

Função:

- Recebe dados canalizados por Nginx, onde múltiplos dispositivos Pico via HTTP podem se beneficiar dessa estratégia.
- Consolida e valida pacotes recebidos, assegurando integridade.
- Executa proxy reverso, roteamento e controle de acesso.
- Processa dados localmente: filtragem, compactação, pró-análise.
- Garante segurança: firewall, autenticação JWT, monitoramento.

- Encaminha os dados para a nuvem por meio de APIs seguras.
- Gerencia alta disponibilidade e resiliência do sistema local.

Interfaces:

- Rede local interna (wi-fi) em mesmo nível que a BDL, assegurando troca de mensagens entre ambas. Etapas na codificação do lwIP registram a BitDogLab na rede wi-fi, de forma que fique apta a se comunicar com outro servidor.

3. Nuvem

Função:

- Fornece meios de trafego criptografado de ponta-a-ponta, entre o servidor local e a nuvem.
- Oculta IPs reais e previne ataques (DDoS, exploits, etc).
- Fornece SSL/TLS para comunicação segura ponta a ponta com certificdado.
- Mantém certificados de forma autônoma para confirmação de autenticidade (devem ser renovados a cada 6 meses – muito comum) sem renovação automática, garantindo 'https', o que demonstra na comunicação entre servidores, cumprir com as exigências mais atuais, seguindo todas as melhores práticas modernas para a Indústria 4.0.
- Otimiza a performance ***EM ESCALA GLOBAL*** do serviço.

Interfaces:

- Rede pública da Internet para a nuvem.
- Rede local via servidor e roteador.
- Responsabilidades muito bem separadas, garantindo baixíssimo acoplamento, o que garante que caso haja interrupção de um serviço, o outro não é afetado.
- Caso haja perda de conexão com a internet pública o banco de dados pode segurar os dados, pois estes residem no servidor, não na BitDogLab.
- Altíssima coesão garante que todo assunto convertido em funcionalidade do software, bem como da estrutura, possui: começo, meio e fim em uma mesma unidade. Também, nas periferias há apenas unidades correlatas e independentes, guardadas as proporções.
- Unidades individuais têm tratamento individual e static.

2 – Descrição das funcionalidades.

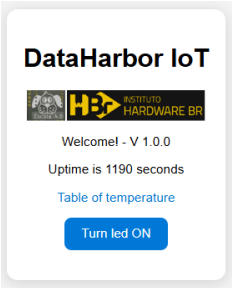

Figura 6: Descrição das Funcionalidades.

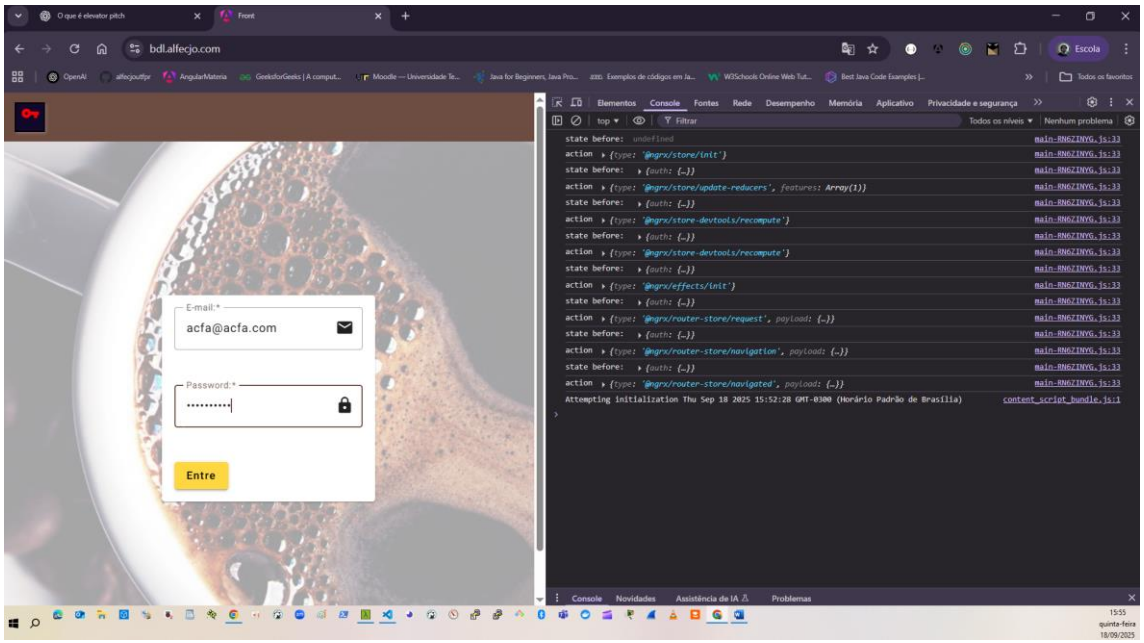

Figura 7: Tela de Abertura.

Uptime Fornece o tempo em segundos desde a inicialização do aplicativo no navegador, ou seja, desde a primeira chamada feita pelo cliente.
Trata-se de uma forma dinâmica de demonstrar **controle de tempo**, recurso essencial em aplicações modernas para monitoramento, logs e sincronização de eventos.

Table of temperature é um link que conduz a uma nova homepage, na qual são exibidas as leituras da **temperatura do chip em tempo real**, juntamente com a **média aritmética** dos valores coletados.
Esse recurso demonstra na prática a **coleta, armazenamento e transformação de dados**, requisitos fundamentais em projetos IoT mais exigentes. A capacidade de sintetizar informações permite **tomada de decisão baseada em dados** e até a criação de mecanismos de **inteligência embutida**.

Turn led ON trata-se de demonstração prática de **manipulação e controle de cargas em tempo real**, simulando o acionamento de dispositivos físicos. Esse recurso é obrigatório em projetos IoT voltados a **controle e automação**, servindo como exemplo de integração entre software e hardware

Todas essas funcionalidades tornam o projeto não apenas um exemplo acadêmico, mas também um **protótipo funcional de IoT**, capaz de demonstrar conceitos de monitoramento, processamento de dados e automação — pilares fundamentais da Indústria 4.0 e de soluções inteligentes conectadas.

3 – Definição das Variáveis e Constantes do Sistema.
Em uma aplicação dessa envergadura são muitas as variáveis e constantes que interagem, contudo, apresentaremos as mais importantes e que demonstram valores tangíveis, já com suas explicações em forma de comentários.

```
18   static float temp_sum = 0.0f;
19   static int temp_count = 0;
20   static float temp_avg = 0.0f; // to store the average temperature
```

```
24   static absolute_time_t wifi_connected_time; // to track when the WiFi connection was established
25   static bool led_on = false;
```

```c
static tCGI cgi_handlers[] = {
    {"/", cgi_handler_test},
    {"/index.shtml", cgi_handler_test},
};
```

```c
#define LED_STATE_BUFSIZE 4
static void *current_connection;
```

Em projetos microcontrolados, onde os recursos devem ser gerenciados e otimizados ao máximo, quanto mais baixo nível no código, isso se reflete em menor consumo desses mesmos recursos. Nesta linha, foi feito o uso intensivo de tecnologias como: ***SSI-Server Side Include*** e ***CGI- Common Gateway Interface***, as quais são recomendadas e muito presentes em linguagem C e comunicação http.

4 – Fluxograma.

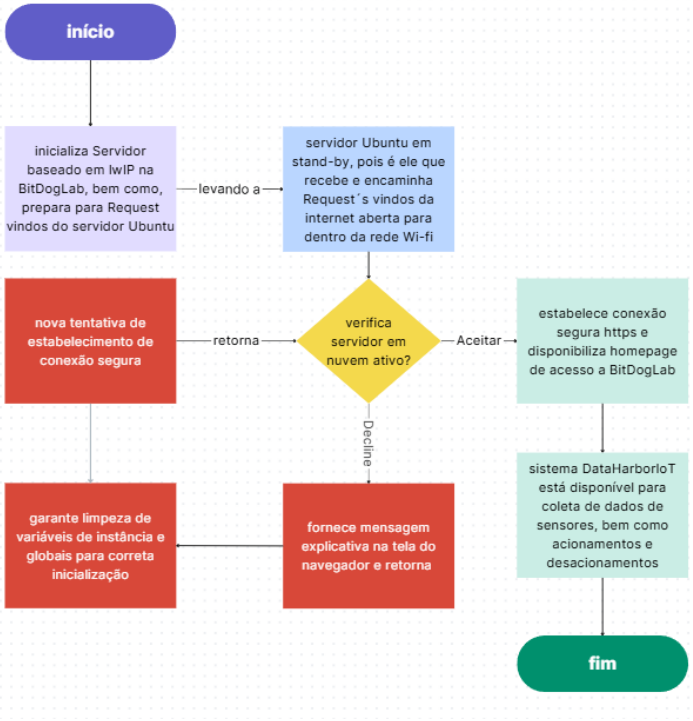

Figura 8: Fluxograma.

7 – Estrutura e formato dos dados apresentados na aplicação em alto nível.

- ✓ ADC (Analogic Digital-Converter) é um sensor de temperatura gera uma tensão proporcional à temperatura.
- ✓ Essa tensão é enviada internamente ao **ADC4**, que converte o valor analógico em um valor digital de 12 bits (0–4095).
- ✓ ***ADC** Fórmula de Conversão (datasheet do RP2040):*
  - ***T (°C) = 27 - (V_sense - 0.706) / 0.001721***
- ✓ ***No C/C++, o pico-sdk** já fornece funções auxiliares para simplificar isso:*
  - *adc_init()*
  - *adc_set_temp_sensor_enabled(true)*
  - *adc_select_input(4)*
  - *adc_read()*

8 – Organização da memória.

- ➢ Exemplo simplificado em termos de memória:
  - **Flash (XIP)**: contém o código da função read_temp_sensor().

- **Periférico ADC (endereços mapeados)**: contém os registradores que o CPU acessa para disparar e ler a conversão.
- **SRAM**: armazena a variável temp_celsius com o valor calculado.
- **Registradores internos da CPU**: usados momentaneamente nos cálculos (por exemplo, multiplicações/divisões para converter o valor do ADC em graus).

9 –Utilização de GPIO na questão de carga (liga/desliga).

- **GPIO25** é um pino da matriz de I/O do **RP2040** (o microcontrolador da Pico W).
- Ele está conectado fisicamente ao LED embutido na placa (geralmente verde).
- O GPIO25 pode operar como **entrada** ou **saída**. No caso do LED, configuramos como **saída digital**.
- O SDK da Pico (pico-sdk) fornece funções para manipular GPIO:
  - gpio_init(25); → inicializa o GPIO25.
  - gpio_set_dir(25, GPIO_OUT); → define como saída.
  - gpio_put(25, 1); → coloca nível lógico **alto** no pino (LED acende).
  - gpio_put(25, 0); → coloca nível lógico **baixo** (LED apaga).
  - Essas funções de alto nível chamam, internamente, **escritas em registradores de hardware**.

d) Execução do projeto

1 – Escolha da Base (Esqueleto)

- No decorrer de todo o Embarcatech foram muitos os aprendizados, sendo assim, fica praticamente impossível ancorar a responsabilidade em uma única escolha, contudo, foi observado com maior rigor, em seguida selecionado o exemplo de servidor HTTP disponibilizado no SDK do Pico W.
- Esse exemplo forneceu a estrutura inicial de conectividade Wi-Fi, entendimento crucial a lógica que seria utilizada, bem como, atendimento a requisições web, servindo como ponto de partida sólido.

2– Estudo e Compreensão do Código Existente.

- Estudo do fluxo principal: inicialização do hardware, configuração da pilha TCP/IP, criação do servidor HTTP e tratamento básico de requisições.
- Identificação dos pontos de extensão para adicionar novas funcionalidades (rotas, interações com hardware, sensores).

3– Análise e Concepção do Código Existente.

- Todo fluxo principal precisou ser adaptado as novas exigências, tanto do hardware, como adaptação do servidor seguindo especificação de projeto em direção ao objetivo a ser alcançado.

4– Incremento Iterativo.

- Testes incrementais: cada nova funcionalidade (ex.: ligar LED, ler temperatura) foi implementada, testada individualmente e só então integrada ao fluxo principal.
- Ajustes progressivos na organização da memória (stack, buffers de rede, armazenamento das páginas Angular).

5– Integração Hardware–Software.

- Conexão entre a camada física (GPIO, ADC, Wi-Fi) e a camada lógica (servidor HTTP e respostas às requisições).
- Garantia de que cada recurso de hardware tivesse sua correspondente “porta de entrada” pelo software via HTTP.

6– Validação Final.

- Teste do sistema ponto-a-ponto: acesso ao Pico W por navegador, verificação da resposta das rotas (ex.: /led/on, /led/off, /temp).
- Ajustes de usabilidade e performance para garantir estabilidade e confiabilidade da solução.
- Testes unitários individuais para verificar o sistema e seu funcionamento em partes.
- Comprovada aceitação foram implementados testes funcionais, não funcionais, manuais e automatizados e de integração, assegurando que todas as partes funcionam juntas e separadas.
- Teste de deploy, primeiro apenas containers individuais e em seguida, utilização do Docker Compose, escalando tudo para a nuvem, oferecendo alta disponibilidade aos usuários.
- Essa metodologia é **incremental e adaptativa**: o ponto focal principal foi “pico_w/wifi/tcp_server/picow_tcp_server.c”, bem como todas as suas dependências, assim, não se reescreveu tudo do zero, mas foi evoluindo a partir de um esqueleto já funcional, adaptando-o às necessidades reais de projeto. Nessa abordagem o esqueleto é mantido, contudo, quase sempre as funcionalidades existentes não condizem às esperadas pelo projeto atual. Sendo assim, foi necessário escrever toda a lógica visando a cobertura das exigências, as quais devem ser comtempladas. Ao final, tantas são as mudanças que não há mais identidade entre os projetos. Guardadas as proporções descritas como esqueleto.

6– Principais desafios de projeto.

O desenvolvimento de uma solução embarcada utilizando o **Raspberry Pi Pico W (BitDogLab)** conectado a um ambiente de computação em nuvem, por meio de um servidor intermediário baseado em **Ubuntu 25.04.0 LTS (64-bit)**, impôs uma série de desafios técnicos que precisaram ser superados ao longo do projeto.

A seguir, são apresentados os principais pontos críticos enfrentados:

1. Adaptação *do Servidor HTTP baseado em lwIP:*

O primeiro desafio consistiu em trabalhar com o **esqueleto do servidor HTTP provido pela pilha lwIP**. Essa pilha, embora bastante eficiente para sistemas embarcados de recursos limitados, exige grande esforço de configuração e integração manual e o código não é nada amigável. Então, foi necessário estudar uma forma que contemplasse maior eficiência em atacar o problema.

Foi necessário:

- Ajustar as rotinas do servidor para manipular requisições específicas de leitura e escrita.
- Criar handlers capazes de **interpretar comandos do cliente** e refletir tais comandos em operações diretas de hardware, como o acionamento de GPIOs.

- Garantir o funcionamento do servidor em condições de restrição de memória, visto que o Pico W possui recursos limitados de RAM e processamento.
- Ajustar a todo momento a sincronização, de modo que não se perdesse o timer nas interações. Haja visto que o resultado esperado propõe a comunicação entre um jato supersônico e um drone, simbolicamente falando.

2. Controle de GPIO e Interação com Periféricos

Outro desafio foi a **integração direta com os periféricos do microcontrolador**.

O projeto demandou:

- A implementação do controle digital sobre o **GPIO25**, responsável por ligar e desligar o LED.
- A realização de **leituras periódicas do ADC no canal 4 (são dez leituras para obter a média aritmética)**, o qual está internamente vinculado ao **sensor de temperatura do chip**.
- A sincronização entre o ciclo de requisições HTTP e as respostas rápidas do hardware, de forma que os usuários pudessem perceber o comportamento quase em tempo real ao interagir via interface web.

3. Conexão entre o Pico W e o Servidor Ubuntu

A arquitetura projetada não previa uma conexão direta do dispositivo embarcado à nuvem, mas sim por meio de um **servidor intermediário Ubuntu 25.04.0 LTS (64-bit)**, que atua como **proxy reverso**.

Esse arranjo trouxe os seguintes desafios:

- Configuração de protocolos de comunicação confiáveis entre o Pico W e o servidor intermediário, utilizando **HTTP sem criptografia** (limitado pelo dispositivo embarcado).
- Garantir baixa latência e resiliência da comunicação, de forma que o servidor proxy pudesse repassar comandos e respostas de maneira transparente ao usuário final.
- Implementação de mecanismos de reconexão e tolerância a falhas, mitigando instabilidades de rede local ou desconexões temporárias do dispositivo.

4. Segurança e Requisitos de Conectividade com a Nuvem

A segurança representou um dos maiores desafios. Embora o **Pico W com lwIP ofereça suporte a HTTPS de forma eficiente**, foi necessário transferir essa responsabilidade ao servidor Ubuntu, diminuindo a complexidade do firmware:

- Mantém a conexão segura (**HTTPS- SSL/TLS**) com a nuvem.
- Desempenha o papel de **gateway seguro**, isolando o dispositivo embarcado de exposições diretas à Internet pública.
- Garante que todas as requisições dos usuários finais sejam autenticadas e trafeguem por um canal criptografado até a nuvem, onde são redirecionadas para **interação**.

Essa separação de papéis aumentou a robustez do sistema, mas demandou um cuidado adicional com a configuração do proxy reverso e dos certificados digitais.

5. Experiência do Usuário Final e Interação com a Aplicação

Por fim, outro desafio fundamental foi alinhar os aspectos técnicos da solução com uma **experiência de uso simples e intuitiva**. O sistema precisava:

- Permitir ao usuário final **ligar/desligar o LED conectado ao GPIO25** e **visualizar a temperatura interna do chip** através de páginas web hospedadas na nuvem.
- Manter a abstração da complexidade arquitetural, de modo que o usuário interaja apenas com interfaces HTTPS amigáveis, sem necessidade de conhecer os detalhes do proxy ou do microcontrolador.
- Assegurar responsividade, mesmo diante das limitações do hardware embarcado e das múltiplas camadas de comunicação.

7– Link do projeto.
A solução embarcada utilizando o **Raspberry Pi Pico W na forma da BitDogLab encontra-se:**

https://github.com/EmbarcaTech-2025/projeto-final-antonio_almeida.git

Agradecimentos

Gostaria de expressar meus sinceros agradecimentos ao curso **Embarcatech**, cujo enfoque prático na construção de software em **linguagem C** e no desenvolvimento de hardware voltado ao **Raspberry Pi Pico**, por meio da plataforma **BitDogLab**, foi fundamental para a realização deste projeto.

Agradeço a todos os professores e monitores que, com dedicação e paciência, compartilharam seus conhecimentos, esclareceram dúvidas e ofereceram suporte técnico em todas as etapas do aprendizado. Em especial, registro minha profunda gratidão ao **Professor Fruett**, que dedicou grande parte do seu tempo, experiência e atenção ao acompanhamento do desenvolvimento dos alunos, orientando com clareza e incentivo, contribuindo de forma decisiva para o meu crescimento acadêmico e profissional.

A todos que direta ou indiretamente colaboraram para que este trabalho se tornasse possível, meu sincero reconhecimento e agradecimento.

e) Referência Bibliográficas

1 - https://www.raspberrypi.com/documentation/microcontrollers/pico-series.html#pico-1-family

2 - https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf

3 - https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf

4- https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf

5 - https://datasheets.raspberrypi.com/picow/connecting-to-the-internet-with-pico-w.pdf

6 - **Network Programmability and Automation, 2nd Edition.**
By **Matt Oswalt**, **Christian Adell**, **Scott S. Lowe**, **Jason Edelman**

7 - **TCP/IP Illustrated, Volume 1: The Protocols, 2nd Edition**
By **Kevin R. Fall**, **W. Richard Stevens**

8- **How Linux Works, 3rd Edition**
By **Brian Ward**

9 - **The Linux Command Line, 2nd Edition**
By **William E. Shotts**

10 - **NGINX HTTP Server - Fifth Edition**
By **Gabriel Ouiran**, **Nedelcu**, **Martin Bjerretoft Fjordvald**