INSTITUTO HARDWARE BR – EMBARCATECH

Monitoramento e Assistência Homecare

Aluno:

Felipe Leme Correa da Silva
Vitor Gomes de Souza

Campinas
2025

SUMÁRIO

1. Situação Problema

Com o aumento da população idosa e de pacientes com doenças cardíacas e respiratórias, o uso de tecnologias de monitoramento torna-se essencial para garantir segurança e qualidade de vida no atendimento domiciliar (Homecare).

Monitorar sinais vitais como batimentos cardíacos e oxigenação do sangue em tempo real permite prevenir complicações e possibilita uma resposta rápida em emergências. Além disso, a detecção de quedas e a automação de alertas são medidas fundamentais para preservar a saúde e a autonomia do paciente.

Este projeto propõe uma solução IoT modular e eficiente para o monitoramento remoto da saúde. O sistema coleta sinais vitais em tempo real, detecta quedas ou desmaios e emite alarmes. Todos os registros ficam armazenados em arquivos CSV, permitindo consultas futuras pela equipe médica responsável.

O paciente dispõe de um tempo de segurança de até 120 segundos para cancelar o alarme (através de um botão físico na BitDogLab). Caso não haja resposta, o sistema aciona o protocolo de emergência.

A solução é baseada em duas unidades principais:

- **Pulseira de Monitoramento:** dispositivo vestível com Raspberry Pi Pico W (BitDogLab), responsável pela coleta dos sinais vitais e detecção de quedas. Possui alertas visuais (LED RGB) e sonoros (Buzzer), além de publicar os dados no broker via protocolo MQTT.

- **Servidor Local / Broker MQTT:** Raspberry Pi 4 atuando como broker, recebendo os dados da pulseira, armazenando registros em arquivos CSV/JSON e servindo como ponte de comunicação para análise médica.

2. Requisitos

Por se tratar de um sistema voltado à integridade do paciente e ao monitoramento contínuo, foram definidos os seguintes requisitos:

- Medição contínua de batimentos cardíacos e saturação de oxigênio (sensor MAX30100);
- Detecção de queda e desmaios (sensor MPU6050);
- Lembrete de hidratação (local): temporizador no firmware que sinaliza periodicamente o usuário via LED RGB e buzzer.
- Feedback visual (LED RGB da BitDogLab);
- Feedback sonoro (Buzzer da BitDogLab);
- Interação direta do usuário via botões de emergência e cancelamento de queda;
- Gerenciamento multitarefa utilizando FreeRTOS;
- Emissão de alarmes em caso de quedas ou valores anormais de BPM e SpO_2;
- Armazenamento estruturado de dados em formato CSV e JSON;
- Comunicação via protocolo MQTT, com broker local embarcado no Raspberry Pi 4.

3. Lista de Materiais

Item	Quantidade
BitDogLab (Raspberry Pi Pico W - RP2040)	1
MAX30100	1
MPU6050	1
Raspberry Pi 4	1
Fonte de Alimentação DC +5V	2

Tabela 01: Lista de materiais

Utilizaremos os componentes internos da BitDogLab para simular a interação do usuário, além das visualizações visuais e sonoras.

A interação direta do usuário, para evitar que o sistema detecte falsos positivos de quedas ou desmaios, ocorrerá por meio do botão físico presente na BitDogLab.

Além disso, contará com um botão de emergência, para situações que ocorram devido a alguma urgência, sendo necessário o acionamento imediato da equipe médica.

Os sensores serão conectados via barramento I2C utilizando os conectores da BitDogLab, garantindo integridade na comunicação e nas leituras.

4. Arquitetura do Hardware

Utilizando a BitDogLab, os sensores externos e o microprocessador Raspberry Pi 4, elaboramos a arquitetura do sistema de forma modular, definindo como cada componente interage e evidenciando de maneira clara suas interconexões e processos.

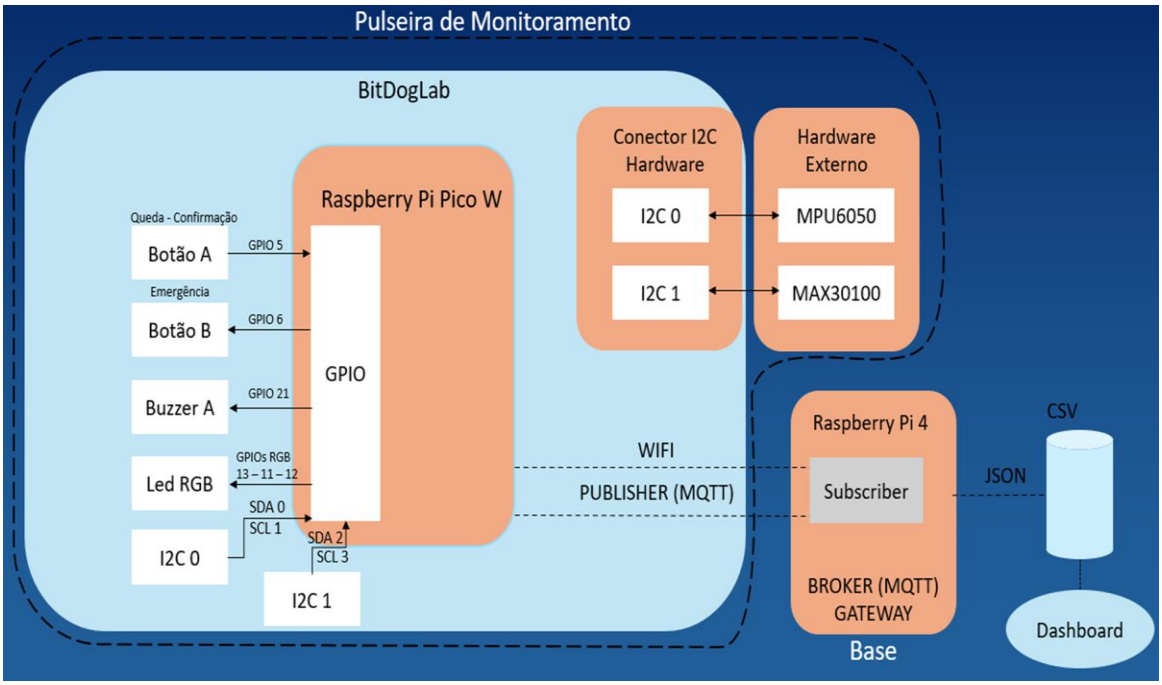

Foto 01: Arquitetura de Hardware

O esquema acima aborda os seguintes tópicos:

Pulseira de Monitoramento (BitDogLab + Hardware):

- Botão A: Permite interação manual do usuário, como cancelar alarmes de queda.
- Botão B: Botão de emergência para eventuais problemas relacionados a condição médica do usuário;
- Buzzer A: Emite alertas sonoros em caso de anomalias ou quedas detectadas.
- LED RGB: Fornece feedback visual (ex.: vermelho = alerta).
- GPIO: Faz a interface entre os periféricos internos (botão, buzzer, OLED, LED RGB).
- I2C 0 e I2C 1: Canais de comunicação para os sensores externos.
- Lembrete de Hidratação (timer local):

- Sem hardware adicional. Usa os periféricos da BitDogLab (LED RGB e buzzer).
- Sinalização atual: LED azul piscante + bip curto (padrão humano-amigável).
- Convivência com segurança: quando ALERTA_QUEDA está ativo, o lembrete fica suspenso e é retomado após o fechamento do evento.

1. Sensores Externos (via I2C):

- MPU6050: Detecta movimentos bruscos ou quedas usando acelerômetro e giroscópio.

- MAX30100: Mede frequência cardíaca (BPM) e saturação de oxigênio (SpO_2).

- Conector I2C Hardware: Faz a ponte entre a BitDogLab e os sensores.

2. Comunicação via Wi-Fi e MQTT:

- A BitDogLab atua como Publisher MQTT, enviando leituras de BPM, SpO_2 e eventos de queda e emergência para a Raspberry Pi 4.

- Essa comunicação ocorre pela rede Wi-Fi local, com mensagens em tópicos específicos (ex.: homecare/bracelet/pico-homecare-01/vitals; {"ts_ms", "bpm", "spo2"}).

3. Microprocessador Raspberry Pi 4:

- Broker MQTT: Recebe mensagens publicadas pela BitDogLab.

- Subscriber: Armazena os dados em formatos JSON/CSV, gerando histórico para análise.

4. Dashboard:

- O Dashboard Médico acessa os dados do Mosquitto (broker) para mostrar informações em tempo real e relatórios históricos.

5. Metodologia

A metodologia aplicada no desenvolvimento da pulseira Homecare envolveu tanto a programação embarcada quanto a configuração de serviços no servidor central. O projeto foi dividido nas seguintes etapas:

1. Desenvolvimento dos drivers dos sensores e periféricos;
2. Criação de tarefas FreeRTOS para leitura de sinais vitais e detecção de quedas;
3. Implementação do protocolo MQTT no Raspberry Pi Pico W para envio dos dados;
4. Configuração de broker e subscriber no Raspberry Pi 4. Implementação de um logger Python, integrado ao systemd, para registro em CSV;
5. Desenvolvimento de dashboard para leitura e análise dos dados armazenados.

Etapa 1 – Desenvolvimento dos drivers

- MAX30100: rotinas I²C para inicialização, configuração de registradores e leitura dos sinais de BPM e SpO_2. Utilizou-se o canal I²C1 para aquisição.
- MPU6050: driver para leitura de aceleração e giroscópio, incluindo rotina de calibração inicial e aplicação de filtros digitais (média móvel) para reduzir ruídos.
- LED RGB e Buzzer: configurados como periféricos GPIO/PWM para fornecer feedback ao usuário.
- Botões: configurados como entradas digitais. O botão de queda atua como cancelamento de alarmes; o botão de emergência permite acionar diretamente a equipe médica.

Cada driver foi modularizado em arquivos separados (*.c e *.h), garantindo manutenção simplificada, reuso do código e flexibilidade para substituição de sensores.

Etapa 2 – Criação das Tarefas FreeRTOS

Nesta etapa, estruturamos o firmware do Raspberry Pi Pico W em tarefas concorrentes com particionamento por responsabilidade e comunicação via filas, semáforos e grupos de eventos. O objetivo é garantir:

- Leitura estável dos sensores;
- Publicação confiável via MQTT;
- Interação responsiva com o usuário (LED RGB e botões).

Etapa 3 — Implementação do MQTT no Raspberry Pi Pico W

Nesta etapa integramos a conectividade Wi-Fi e o cliente MQTT ao firmware do Pico W. A comunicação segue um design robusto com reconexão exponencial (com jitter), publicação periódica e sob evento, e formatação JSON consistente entre tópicos. O Wi-Fi é inicializado antes do stack MQTT; as publicações só são liberadas após a sinalização de semMqttReady (primeira conexão estabelecida).

Nota: após a primeira conexão, a MqttTask publica apenas se mqtt_is_connected() estiver verdadeiro. Em reconexões, não "des-sinalizamos" o semáforo; usamos o estado do cliente para decidir publicar ou enfileirar.

3.1. Tópicos e Esquema de Mensagens

- Prefixo: homecare/bracelet/<device-id>/...
    - vitals → {"ts_ms", "bpm", "spo2"};
    - event/emergency → {"ts_ms", "src":"fall|button"};
    - status → {"ts_ms", "wifi", "mqtt", "uptime"};
- Device-id usado nos testes: pico-homecare-01.

Boa prática: construir os tópicos dinamicamente com base em <device-id> (evita hard-code e facilita replicar firmware para outros dispositivos).

3.2. QoS, Retain e Políticas

- QoS 0 para vitals (telemetria frequente, tolera perda pontual).
- QoS 1 para event/emergency (garante pelo menos uma vez).
- status com retain = true para facilitar descoberta do dispositivo e último estado conhecido.

- LWT (Last Will & Testament): publicar em status uma payload {"ts_ms":<now>,"wifi":0,"mqtt":0,"uptime":<last>} com retain=true quando o cliente cair inesperadamente (detectado pelo broker).

3.3. Ciclo de Conexão e Reconexão

- Backoff exponencial com jitter: 1s, 2s, 4s, … até 30s (±10% aleatório).
- Heartbeat de status: a cada 30 s.
- Buffer local limitado para eventos quando offline (ex.: ring buffer de 16 entradas).
- Keepalive sugerido: 30–60 s; clean session = true para protótipo (pode evoluir para false + sessão persistente no futuro).

3.4. Esqueleto do Cliente (C, pseudocódigo)

- Dependências ilustrativas: pico/cyw43, cliente MQTT (ex.: mqtt_async ou paho portado), cJSON (opcional).
- Dica: ao usar snprintf com float, habilite printf de float no link (-Wl,-u,_printf_float) ou use inteiros escalados.

```
// ---------- Config ----------
static const char* DEV_ID = "pico-homecare-01";

// Monta tópicos dinamicamente com base no device-id
static char TOP_VITALS[96];
static char TOP_EVENT[128];
static char TOP_STATUS[96];

static inline void build_topics(void){
  snprintf(TOP_VITALS, sizeof(TOP_VITALS), "homecare/bracelet/%s/vitals",
DEV_ID);
  snprintf(TOP_EVENT,  sizeof(TOP_EVENT),  "homecare/bracelet/%s/event/emergen
cy", DEV_ID);
  snprintf(TOP_STATUS, sizeof(TOP_STATUS), "homecare/bracelet/%s/status",
DEV_ID);
}

// ---------- APIs (stubs) ----------
void wifi_init_and_connect(const char* ssid, const char* pass);
bool mqtt_connect_with_backoff(void);           // aplica backoff + jitter
bool mqtt_is_connected(void);
bool mqtt_publish_json(const char* topic, const char* json, int qos, bool
retain);
void mqtt_set_lwt(const char* topic, const char* json, int qos, bool retain);
```

```
uint32_t ts_ms(void);                              // monotônico (ex.:
to_ms_since_boot)
uint32_t uptime(void);                             // em segundos
int wifi_ok(void);                                 // 0|1

// Sinalização da 1ª conexão
SemaphoreHandle_t semMqttReady;

// ---------- Inicialização ----------
void net_init(void){
  build_topics(); // monta strings de tópicos uma vez
  wifi_init_and_connect(WIFI_SSID, WIFI_PASS);

  // Configura LWT antes de conectar
  char lwt[128];
  snprintf(lwt, sizeof(lwt),
    "{"
      "\"ts_ms\":%lu,"
      "\"wifi\":0,"
      "\"mqtt\":0,"
      "\"uptime\":%lu"
    "}",
    (unsigned long)ts_ms(), (unsigned long)uptime()
  );
  mqtt_set_lwt(TOP_STATUS, lwt, /*qos=*/0, /*retain=*/true);
}

// ---------- Task de supervisão Wi-Fi/MQTT ----------
void wifi_mqtt_task(void* arg){
  net_init();
  for(;;){
    if(!mqtt_is_connected()){
      if(mqtt_connect_with_backoff()){
        xSemaphoreGive(semMqttReady); // libera a primeira vez; reconexões não
precisam dar novamente
        publish_status();             // status imediato pós-conexão
      }
    }
    vTaskDelay(pdMS_TO_TICKS(1000));
  }
}

// ---------- Publicações ----------
typedef struct { uint32_t ts_ms; float bpm, spo2; } vitals_t;

void publish_vitals(const vitals_t* v){
  char json[160];
  // Habilitar printf float no link OU usar inteiros escalados (ex.: bpm*100)
```

```
  snprintf(json, sizeof(json),
    "{"
      "\"ts_ms\":%lu,"
      "\"bpm\":%.2f,"
      "\"spo2\":%.2f"
    "}",
    (unsigned long)v->ts_ms, v->bpm, v->spo2
  );
  if(mqtt_is_connected()){
    mqtt_publish_json(TOP_VITALS, json, /*qos=*/0, /*retain=*/false);
  } else {
    // opcional: enfileirar localmente a última amostra ou descartar
  }
}


void publish_event_emergency(uint32_t ts, const char* src){
  char json[128];
  snprintf(json, sizeof(json),
    "{"
      "\"ts_ms\":%lu,"
      "\"src\":\"%s\""
    "}",
    (unsigned long)ts, src
  );
  if(mqtt_is_connected()){
    mqtt_publish_json(TOP_EVENT, json, /*qos=*/1, /*retain=*/false);
  } else {
    // enfileirar em ring buffer (cap=16) para publicar ao reconectar
  }
}

void publish_status(void){
  char json[160];
  snprintf(json, sizeof(json),
    "{"
      "\"ts_ms\":%lu,"
      "\"wifi\":%d,"
      "\"mqtt\":%d,"
      "\"uptime\":%lu"
    "}",
    (unsigned long)ts_ms(), wifi_ok(), mqtt_is_connected(), (unsigned
long)uptime()
  );
  mqtt_publish_json(TOP_STATUS, json, /*qos=*/0, /*retain=*/true);
}
```

MqttTask (loop de publicação)

- Publica vitals a cada 1 s (ou período definido).
- Drena qEvents e publica event/emergency imediatamente quando houver.

```
void MqttTask(void* arg){
  // Espera a primeira conexão
  xSemaphoreTake(semMqttReady, portMAX_DELAY);

  TickType_t last = xTaskGetTickCount();
  for(;;){
    vTaskDelayUntil(&last, pdMS_TO_TICKS(1000)); // período de vitals

    // Vitals (última amostra disponível)
    vitals_t v;
    if(xQueuePeek(qVitals, &v, 0)){
      publish_vitals(&v);
    }

    // Events (drain)
    event_t e;
    while(xQueueReceive(qEvents, &e, 0)){
      if(strcmp(e.tipo, "fall") == 0 || strcmp(e.tipo, "button") == 0){
        publish_event_emergency(e.ts_ms, e.src);
      }
    }

    // Heartbeat de status (30 s)
    static uint32_t last_status = 0;
    if(ts_ms() - last_status >= 30000){
      publish_status();
      last_status = ts_ms();
    }
  }
}
```

3.5. Provisionamento de Wi-Fi e Credenciais

- Protótipo: credenciais embutidas (WIFI_SSID/WIFI_PASS).
- Alternativa: leitura de /flash/wifi.cfg (NVS/LittleFS) para facilitar campo.
- Segurança (futuro próximo): TLS com CA do broker + usuário/senha MQTT (Mosquitto).

- clientId: use algo único por dispositivo, ex.: homecare-<DEV_ID> (evita “kicks” por duplicidade).

3.6. Tratamento de Erros

- Timeouts de socket e watchdog no loop MQTT.
- Re-init do driver Wi-Fi após falhas consecutivas (ex.: 5 tentativas).
- Contadores de estatística (expostos em status): conn_attempts, restarts, pub_fail, last_rc.
- Proteção de payload: validar limites (ex.: bpm negativo, spo2 > 100) antes de publicar.

Etapa 4 — Broker MQTT e Logger CSV no Raspberry Pi 4 (Revisado)

No Raspberry Pi 4, utilizamos o Mosquitto como broker MQTT e um serviço Python (subscriber/logger) gerenciado pelo systemd para persistir dados em CSV. O logger assina os tópicos de interesse e grava linhas normalizadas com timestamp (ISO e ms), origem (device_id) e payload decodificado. (Sem detalhes de dashboard nesta etapa.)

4.1 Instalação do Mosquitto (Broker)

```
# RPi4 (Debian/Raspberry Pi OS)
sudo apt update
sudo apt install -y mosquitto mosquitto-clients
sudo systemctl enable mosquitto
sudo systemctl start mosquitto

# Teste rápido
mosquitto_sub -t 'homecare/#' -v &
mosquitto_pub -t 'homecare/test' -m 'hello'
```

4.2 Configuração essencial do Mosquitto

/etc/mosquitto/conf.d/homecare.conf (exemplo simples para teste):

```
listener 1883 0.0.0.0
allow_anonymous true
persistence true
persistence_location /var/lib/mosquitto/
```

4.3 Estrutura de diretórios do logger

```
/opt/homecare/
├── homecare_logger.py
├── data/
│   ├── vitals-YYYY-MM-DD.csv
│   ├── events-YYYY-MM-DD.csv
│   └── status-YYYY-MM-DD.csv
└── logs/
    └── homecare-logger.log
```

4.4 Logger Python (subscriber → CSV)

- Assina três tópicos usando wildcard para aceitar múltiplos devices.
- Escreve CSVs separados por tipo com rotação diária no nome do arquivo.
- Adiciona device_id extraído do tópico.

```
#!/usr/bin/env python3
# /opt/homecare/homecare_logger.py
import json, os, time, csv, re, logging
from datetime import datetime
import paho.mqtt.client as mqtt

DATA_DIR = '/opt/homecare/data'
LOG_DIR  = '/opt/homecare/logs'
BROKER   = 'localhost'
PORT     = 1883
KEEPALIVE = 60

# Tópicos com wildcard +: homecare/bracelet/<device-id>/...
TOP_VITALS = 'homecare/bracelet/+/vitals'
TOP_EVENT  = 'homecare/bracelet/+/event/emergency'
TOP_STATUS = 'homecare/bracelet/+/status'

os.makedirs(DATA_DIR, exist_ok=True)
os.makedirs(LOG_DIR, exist_ok=True)

logging.basicConfig(
    filename=os.path.join(LOG_DIR, 'homecare-logger.log'),
    level=logging.INFO,
    format='%(asctime)sZ %(levelname)s %(message)s',
    datefmt='%Y-%m-%dT%H:%M:%S'
)

# Arquivos ativos por dia
open_files = {}
```

```
def daily_path(kind, day):
    return os.path.join(DATA_DIR, f'{kind}-{day}.csv')

def ensure_csv(kind, header):
    day = time.strftime('%Y-%m-%d')
    key = (kind, day)
    if key not in open_files:
        path = daily_path(kind, day)
        exists = os.path.exists(path)
        f = open(path, 'a', newline='')
        w = csv.writer(f)
        if not exists:
            w.writerow(header)
        open_files[key] = (f, w)
    return open_files[key]

def parse_device_id(topic):
    # homecare/bracelet/<device-id>/...
    parts = topic.split('/')
    return parts[2] if len(parts) > 2 else 'unknown'

def to_ts(ts_ms=None):
    if ts_ms is None:
        ts_ms = round(time.time() * 1000)
    ts_iso = datetime.utcfromtimestamp(ts_ms / 1000.0).isoformat() + 'Z'
    return ts_iso, int(ts_ms)

def on_connect(client, userdata, flags, rc):
    subs = [(TOP_VITALS,0), (TOP_EVENT,1), (TOP_STATUS,0)]
    client.subscribe(subs)
    logging.info(f'Connected rc={rc}; subscribed: {subs}')

def on_message(client, userdata, msg):
    try:
        payload = json.loads(msg.payload.decode('utf-8'))
        device_id = parse_device_id(msg.topic)

        # Timestamp robusto (usa ts_ms do payload ou agora)
        ts_ms = payload.get('ts_ms')
        ts_iso, ts_ms = to_ts(ts_ms)

        if msg.topic.endswith('/vitals'):
            f, w = ensure_csv('vitals',
['ts_iso','ts_ms','device_id','bpm','spo2'])
            w.writerow([ts_iso, ts_ms, device_id, payload.get('bpm'),
payload.get('spo2')])
            f.flush()
```

```
        elif msg.topic.endswith('/event/emergency'):
            f, w = ensure_csv('events', ['ts_iso','ts_ms','device_id','src'])
            w.writerow([ts_iso, ts_ms, device_id, payload.get('src')])
            f.flush()

        elif msg.topic.endswith('/status'):
            f, w = ensure_csv('status',
['ts_iso','ts_ms','device_id','wifi','mqtt','uptime'])
            w.writerow([ts_iso, ts_ms, device_id, payload.get('wifi'),
payload.get('mqtt'), payload.get('uptime')])
            f.flush()

    except Exception as e:
        logging.exception(f'Error handling message on {msg.topic}: {e}')

def main():
    c = mqtt.Client(client_id='homecare-logger', clean_session=True,
protocol=mqtt.MQTTv311)
    c.on_connect = on_connect
    c.on_message = on_message

    # Backoff de reconexão (1 → 30 s)
    c.reconnect_delay_set(min_delay=1, max_delay=30)

    c.connect(BROKER, PORT, keepalive=KEEPALIVE)
    try:
        c.loop_forever()
    finally:
        # Fecha arquivos abertos ao sair
        for (kind, day), (f, _) in list(open_files.items()):
            f.close()

if __name__ == '__main__':
    main()
```

4.5 Serviço systemd

Serviço para inicializar o logger no boot e reiniciar em falhas.

- /etc/systemd/system/homecare-logger.service:

```
[Unit]
Description=Homecare MQTT CSV Logger
After=network-online.target mosquitto.service
Wants=network-online.target

[Service]
Type=simple
User=pi
Group=pi
ExecStart=/usr/bin/python3 -u /opt/homecare/homecare_logger.py
Restart=always
RestartSec=2
WorkingDirectory=/opt/homecare
# Logs em arquivo (já configurado pelo script)
StandardOutput=append:/opt/homecare/logs/homecare-logger.log
StandardError=append:/opt/homecare/logs/homecare-logger.log

# Endurecimento básico (opcional; comente se der conflito)
NoNewPrivileges=yes
PrivateTmp=yes
ProtectSystem=full
ReadWritePaths=/opt/homecare
AmbientCapabilities=

[Install]
WantedBy=multi-user.target
```

Comandos:

```
sudo systemctl daemon-reload
sudo systemctl enable homecare-logger.service
sudo systemctl start homecare-logger.service
sudo systemctl status homecare-logger.service
```

4.6 Estrutura dos CSVs (exemplo):

- vitals-2025-09-08.csv

```
ts_iso,ts_ms,device_id,bpm,spo2
2025-09-08T19:42:33Z,355363,pico-homecare-01,72.10,97.23
```

- events-2025-09-08.csv

```
ts_iso,ts_ms,device_id,src
2025-09-08T19:42:00Z,354298,pico-homecare-01,button
```

status-2025-09-08.csv

```
ts_iso,ts_ms,device_id,wifi,mqtt,uptime
2025-09-08T19:42:30Z,355000,pico-homecare-01,1,1,123456
```

4.7 Troubleshooting

- Logs do broker: sudo journalctl -u mosquitto -f
- Logs do logger: tail -f /opt/homecare/logs/homecare-logger.log
- Testes pub/sub: mosquitto_pub / mosquitto_sub
- CSV não atualiza: checar permissões em /opt/homecare/data e status do serviço (systemctl status).
- Múltiplos devices: confirme que o Pico publica em homecare/bracelet/<device-id>/...; o logger com + já captura.
- Conectividade local: se publicar de outra máquina, verifique firewall/roteador para a porta 1883.

Estrutura do código

```
HW_CONTROL/
├─ .vscode/
├─ build/
├─ drivers/
│  ├─ max30100.c
│  ├─ mpu6050.c
│  ├─ mqtt.c
│  ├─ rgb_buzzer.c
│  └─ ssd1306.c
├─ FreeRTOS/
├─ inc/
│  ├─ FreeRTOSConfig.h
│  ├─ lwipopts.h
│  ├─ max30100.h
│  ├─ mpu6050.h
│  ├─ mqtt.h
│  ├─ ssd1306.h
│  └─ ssd1306_font.h
├─ src/
│  └─ main.c
├─ .gitignore
├─ CMakeLists.txt
└─ pico_sdk_import.cmake
```

Detalhamento:

- HW_CONTROL/ – raiz do projeto (configs de build e fontes principais).
- .vscode/ – configs do VS Code (tarefas, IntelliSense, etc.).
- build/ – saída do CMake/compilação (gerada; não versionar).
- drivers/ – módulos em C que falam direto com hardware/serviços:
    - max30100.c → leitura do MAX30100 (I²C1), cálculo básico de BPM/SpO_2.
    - mpu6050.c → leitura do MPU6050 (I²C0), calibração/filtragem base.
    - mqtt.c → Wi-Fi/MQTT (conectar, publicar vitals/event/status).
    - rgb_buzzer.c → padrões de LED RGB e buzzer (alertas/feedback).
    - ssd1306.c → driver do OLED (opcional nesta versão).

- FreeRTOS/ – código-fonte do kernel FreeRTOS (se usado como submódulo).
- inc/ – headers públicos do projeto:
    - FreeRTOSConfig.h e lwipopts.h → ajustes do RTOS e da pilha lwIP.
    - max30100.h, mpu6050.h, mqtt.h, ssd1306.h → APIs dos drivers.
    - ssd1306_font.h → fonte/bitmap usado pelo OLED.
- src/ – aplicação:
    - main.c → criação das tarefas (Fall/Vitals/UI/MQTT/Net), software timer do lembrete de água, laços principais.
- CMakeLists.txt – lista fontes, inclui inc/, linka libs do Pico/FreeRTOS/lwIP, habilita printf float.
- pico_sdk_import.cmake – bootstrap do SDK do Pico.
- .gitignore – ignora build/, temporários, etc.

Etapa 5 — Dashboard para Visualização de Dados (Revisado)

O dashboard consome os CSVs gerados pelo logger no Raspberry Pi 4 e apresenta BPM/SpO_2 em tempo quase real, além de eventos e status de conectividade. É simples de manter, roda em qualquer PC com Python 3.x e não depende do broker para exibição.

5.1 Estrutura de dados (entrada)

- vitals.csv → séries temporais de BPM e SpO_2 com ts_iso e ts_ms.
- events.csv → registros de queda e botão (timestamp e src).
- status.csv → Wi-Fi/MQTT (0/1) e uptime.

5.2 Ferramentas

- Python 3.x
- Pandas (leitura/filtragem de CSV)
- Matplotlib (gráficos de séries)
- Tkinter (GUI desktop simples) ou Dash/Plotly (web) — opcionais

5.3 Código do dashboard (exemplo enxuto com Matplotlib)

Atualiza a cada 5 s, tenta carregar apenas o arquivo atual e plota BPM/SpO_2; se houver events.csv, marca os eventos no gráfico.

```
import os, glob, time
import pandas as pd
import matplotlib.pyplot as plt

DATA_DIR = "/opt/homecare/data"
VITALS = os.path.join(DATA_DIR, "vitals.csv")     # ou último:
max(glob.glob(f"{DATA_DIR}/vitals-*.csv"))
EVENTS = os.path.join(DATA_DIR, "events.csv")

def latest_csv(pattern, fallback):
    files = glob.glob(pattern)
    return max(files, key=os.path.getmtime) if files else fallback

def load_vitals(path):
    # lê só colunas usadas; converte ts_iso para datetime
    df = pd.read_csv(path, usecols=["ts_iso","bpm","spo2"])
    df["ts_iso"] = pd.to_datetime(df["ts_iso"], errors="coerce")
    df = df.dropna(subset=["ts_iso"]).tail(600)          # ≈ últimos ~10 min
se 1 Hz
    return df

def load_events(path):
    if not os.path.exists(path):
        return None
    de = pd.read_csv(path, usecols=["ts_iso","src"])
    de["ts_iso"] = pd.to_datetime(de["ts_iso"], errors="coerce")
    return de.tail(50)

def main():
    plt.figure("HOMECARE – Vitals")
```

```
    last_vitals_mtime = 0
    last_events_mtime = 0

    while True:
        vitals_path = VITALS
        # descomente se usar rotação diária:
        # vitals_path = latest_csv(f"{DATA_DIR}/vitals-*.csv", VITALS)

        vm = os.path.getmtime(vitals_path) if os.path.exists(vitals_path) else
0
        em = os.path.getmtime(EVENTS) if os.path.exists(EVENTS) else 0

        if vm != last_vitals_mtime or em != last_events_mtime:
            plt.clf()
            if vm:
                df = load_vitals(vitals_path)
                plt.plot(df["ts_iso"], df["bpm"], label="BPM")
                plt.plot(df["ts_iso"], df["spo2"], label="SpO₂")
                plt.legend()
                plt.title("Monitoramento de Sinais Vitais")
                plt.xlabel("Tempo")
                plt.ylabel("Valor")
                # marca última leitura
                if not df.empty:
                    t, bpm, spo2 = df.iloc[-1][["ts_iso","bpm","spo2"]]
                    plt.annotate(f"BPM={bpm:.0f} | SpO₂={spo2:.0f}%",
                                 xy=(t, spo2), xytext=(0.5, 0.9),
textcoords="axes fraction")

            de = load_events(EVENTS)
            if de is not None and not de.empty:
                for _, row in de.iterrows():
                    plt.axvline(row["ts_iso"], linestyle="--", alpha=0.3)
                # legenda extra para eventos recentes
                plt.text(0.01, 0.02, f"Eventos (últimos): {len(de)}",
transform=plt.gca().transAxes)

            plt.xticks(rotation=45)
            plt.tight_layout()

            last_vitals_mtime, last_events_mtime = vm, em

        plt.pause(5)  # atualiza a cada 5 s

if __name__ == "__main__":
    main()
```

Detalhes:

- Usa mtime para evitar reler CSV quando nada mudou.
- tail(600) limita memória e mantém a janela mais útil.
- Marca eventos no gráfico (linhas verticais).
- Funciona igual se os CSVs estiverem crescendo “ao vivo”.

5.4 Próximos incrementos (curtos)

- Alertas visuais quando BPM/SpO_2 fora da faixa (ex.: colorir linha/área; banner “ALERTA”).
- Tela de eventos: lista cronológica com filtro por período e src.
- Modo rotação diária: escolher automaticamente o CSV do dia (já previsto no código).
- Exportação PDF: gerar relatório de período selecionado.
- (Opcional) versão web com Dash/Plotly para multiusuário.

6. Conclusão

Este trabalho consolidou um pipeline completo de Homecare — do wearable à visualização — com foco em confiabilidade, simplicidade e evolução contínua:

- Dispositivo (Pulseira – Pico W): leituras dedicadas em I²C0=MPU6050 (quedas) e I²C1=MAX30100 (BPM/SpO_2), sem mutex, rodando em FreeRTOS com tarefas periódicas, FSM de detecção de queda (impacto → instabilidade → imobilidade → confirmação), parâmetros validados em teste e botão A para cancelamento de alarme. Incluímos ainda o lembrete local de hidratação via software timer (sem MQTT/CSV), pausando automaticamente durante eventos críticos.

- Conectividade: MQTT com tópicos padronizados (vitals, event/emergency, status), QoS adequado a cada fluxo, LWT, e reconexão com backoff + jitter.

- Back-end local (RPi4): Mosquitto como broker e logger Python sob systemd, persistindo CSVs normalizados (e suporte a múltiplos dispositivos) — base simples e auditável para análises.

- Dashboard (CSV-driven): leitura leve com Pandas + Matplotlib, atualização periódica e marcação de eventos, mantendo independência do broker na camada de apresentação.

O resultado é um sistema modular, observável e operacional: dados vitais em 1 Hz, eventos sob demanda, status com retain, e trilha de auditoria em CSV — tudo com baixo acoplamento e fácil manutenção.

Escalabilidade e adaptações (open source)

A arquitetura foi pensada para crescer sem refatorações pesadas:

- Mais dispositivos / multi-usuário: convenção de tópicos com <device-id> já pronta; o logger assina com + e escala de forma natural.
- Novos lembretes e rotinas clínicas: o timer de água abre caminho para medicação (janelas de confirmação, notify/confirm/timeout), persistência local (NVS/LittleFS) e configuração remota via MQTT.
- Segurança e compliance: migração simples para TLS, ACLs e password_file no Mosquitto; trilhas de auditoria já existem pelas mensagens status e registros CSV.
- Sensores e funcionalidades: drivers isolados permitem trocar MAX30100/MPU6050 por equivalentes; inclusão de vibracall, monitor de bateria, e OTA é natural no mesmo esqueleto.
- Análise avançada: fácil acoplar Dash/Plotly ou enviar um espelho para nuvem (AWS IoT/ThingsBoard) mantendo o CSV local; dá para incluir detecção de anomalias/ML no RPi4 sem tocar no firmware.
- Qualidade e comunidade: por ser open source, o projeto favorece issues, PRs, tests (Unity/CMock), CI e releases com artifacts prontos, além de feature flags por #define para variações de campo.

7. Referências

[1] FIOCRUZ. Tecnologias digitais e a Atenção Primária no SUS é tema do terceiro webinário da série 'Transformação digital na Saúde Pública'. Rio de Janeiro, 2024. Disponível em: https://campusvirtual.fiocruz.br/portal/?q=noticia%2F82754. Acesso em: 17 de mai. 2025.

[2] SEBRAE. Inovação e Tecnologia: internet das coisas. Brasília, 2023. Disponível em: https://sebrae.com.br/sites/PortalSebrae/artigos/inovacao-e-tecnologia-internet-das-coisas%2C05d99e665b182410VgnVCM100000b272010aRCRD. Acesso em: 17 de mai. 2025.

[3] ALTUS. Conheça o MQTT, protocolo mais utilizado em aplicações IoT. Porto Alegre, 2023. Disponível em: https://www.altus.com.br/post/194/conheca-o-mqtt-2C-protocolo-mais-utilizado-em-aplicacoes-iot. Acesso em: 17 de mai. 2025.

[4] MINISTÉRIO DA SAÚDE. Saúde da pessoa idosa. Brasília, 2022. Disponível em: https://www.gov.br/saude/pt-br/assuntos/saude-de-a-a-z/s/saude-da-pessoa-idosa. Acesso em: 17 de mai. 2025.

[5] TECNOBLOG. O que é Bluetooth Low Energy (BLE) e quais as vantagens desse tipo de conexão?. São Paulo, 2023. Disponível em: https://tecnoblog.net/responde/o-que-e-bluetooth-low-energy-ble/. Acesso em: 17 de mai. 2025.

[6] ANALOG DEVICES. MAX30100 – Sensor de oximetria de pulso e batimentos cardíacos para aplicações vestíveis. Datasheet, 2014. Disponível em: https://www.analog.com/media/en/technical-documentation/data-sheets/MAX30100.pdf. Acesso em: 17 de mai. 2025.

[7] EMPATICA INC. Manual do Usuário – EmbracePlus. UM-22-Rev 4.0. 2021. Disponível em: https://s3.amazonaws.com/box.empatica.com/manuals/embraceplus/EmbracePlus-UserManual-UM-22-Rev%204.0.pdf. Acesso em: 17 de mai. 2025.

[8] ESPRESSIF SYSTEMS. ESP32-C3 Series Datasheet v2.12. 2020. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf. Acesso em: 17 de mai. 2025.

[9] INVENSENSE / TDK. Especificação do Produto MPU-6000 e MPU-6050. PS-MPU-6000A-00, Rev. 3.4. 2013. Disponível em: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf. Acesso em: 17 de mai. 2025.

8. Apêndice – Dados coletados (MPU6050), cenários:

Pessoa Andando

- MAG (|a|, g): média 1,515, desvio 0,537, p95 2,570, máx 2,990

```
X,Y,Z,MAG,SMOOTH,STATE
0.31,-0.60,1.23,1.41,1.42,0
0.37,-0.56,1.19,1.36,1.41,0
0.44,-0.59,1.09,1.31,1.39,0
0.45,-0.46,1.05,1.23,1.36,0
0.27,-0.38,1.09,1.19,1.32,0
0.14,-0.32,1.25,1.30,1.32,0
0.13,-0.09,1.23,1.24,1.30,0
0.08,-0.21,0.98,1.01,1.24,0
0.07,-0.22,1.01,1.03,1.20,0
0.01,-0.20,1.01,1.03,1.17,0
0.02,-0.21,1.03,1.05,1.16,0
0.06,-0.24,0.97,1.00,1.14,0
0.11,-0.22,0.98,1.01,1.12,0
0.08,-0.16,1.00,1.01,1.11,0
0.10,-0.06,1.02,1.03,1.10,0
0.08,-0.02,1.02,1.03,1.10,0
0.03,-0.12,1.04,1.05,1.10,0
0.01,-0.10,1.01,1.01,1.10,0
0.09,-0.08,0.99,1.00,1.10,0
0.20,-0.03,0.98,1.00,1.10,0
0.24,-0.05,1.01,1.04,1.11,0
0.33,-0.07,1.10,1.15,1.12,0
0.43,-0.12,1.11,1.20,1.14,0
0.43,-0.15,1.15,1.24,1.17,0
0.38,-0.21,1.16,1.25,1.20,0
0.33,-0.28,1.20,1.28,1.24,0
0.32,-0.33,1.21,1.29,1.27,0
0.38,-0.30,1.20,1.28,1.31,0
0.36,-0.35,1.22,1.30,1.35,0
0.34,-0.40,1.23,1.31,1.39,0
```

Pessoa Correndo

- MAG (|a|, g): média 1,101, desvio 0,369, p95 1,589, máx 1.05

```
X,Y,Z,MAG,SMOOTH,STATE
0.05,-0.07,1.01,1.01,1.01,0
0.06,-0.08,1.00,1.00,1.01,0
0.03,-0.09,0.99,0.99,1.00,0
0.02,-0.10,0.99,1.00,1.00,0
0.02,-0.10,0.99,1.00,1.00,0
0.01,-0.10,1.00,1.00,1.00,0
0.03,-0.09,1.01,1.01,1.01,0
0.04,-0.08,1.01,1.01,1.01,0
0.04,-0.07,1.00,1.01,1.01,0
0.04,-0.07,0.99,1.00,1.01,0
0.05,-0.05,0.99,1.00,1.01,0
0.06,-0.04,0.99,1.00,1.02,0
0.06,-0.04,1.00,1.01,1.02,0
0.05,-0.05,1.01,1.01,1.03,0
0.05,-0.06,1.01,1.02,1.04,0
0.05,-0.06,1.01,1.02,1.05,0
0.05,-0.06,1.00,1.01,1.05,0
0.06,-0.07,0.99,1.01,1.05,0
0.06,-0.07,0.99,1.01,1.04,0
0.07,-0.08,0.99,1.01,1.04,0
0.07,-0.08,1.00,1.01,1.03,0
0.07,-0.08,1.00,1.01,1.03,0
0.06,-0.07,1.01,1.01,1.02,0
0.06,-0.06,1.01,1.01,1.02,0
0.05,-0.05,1.02,1.02,1.02,0
0.05,-0.04,1.02,1.02,1.02,0
0.04,-0.04,1.02,1.02,1.02,0
0.04,-0.05,1.01,1.01,1.02,0
0.03,-0.07,1.00,1.01,1.02,0
0.02,-0.09,0.99,1.00,1.02,0
```

Pessoa Sentada

- MAG (|a|, g): média 2,407, desvio 0,296, p95 2,772, máx 2,860.

```
MAG,GYRO,SMOOTH,VAR,STATE
2.23,58.77,2.14,0.41,2
2.18,42.94,2.15,0.41,2
2.06,43.13,2.13,0.41,2
2.07,2.05,2.12,0.37,2
2.09,28.45,2.11,0.32,2
2.18,32.59,2.13,0.28,2
2.12,10.44,2.13,0.22,2
2.09,6.12,2.12,0.16,2
2.11,8.73,2.12,0.12,2
2.12,22.64,2.12,0.08,0
2.10,19.08,2.11,0.06,0
2.11,12.32,2.11,0.06,0
2.13,16.95,2.12,0.06,0
2.13,23.21,2.12,0.06,0
2.12,33.35,2.12,0.06,0
2.11,36.16,2.12,0.05,0
2.18,105.67,2.13,0.04,0
2.13,311.74,2.13,0.04,0
2.26,260.97,2.16,0.04,0
2.38,201.95,2.20,0.09,0
2.40,87.34,2.24,0.13,0
2.44,65.44,2.28,0.17,0
2.52,24.67,2.33,0.22,0
2.66,36.77,2.40,0.28,0
2.70,99.89,2.46,0.34,0
2.70,48.90,2.57,0.46,2
2.86,62.91,2.63,0.51,2
2.77,65.70,2.66,0.54,2
2.75,35.27,2.68,0.56,2
2.76,29.70,2.69,0.58,2
```

Pessoa Sentada e Depois Deitando

- MAG (|a|, g): média 1,724, desvio 0,303, p95 2,085, máx 2,630
- GYRO (|g|, dps): média 85,061, máx 249,060

```
MAG,GYRO,SMOOTH,VAR,STATE
2.57,249.06,1.86,0.46,0
1.06,149.09,1.70,0.43,1
1.83,61.67,1.73,0.36,1
1.70,60.73,1.72,0.31,1
1.64,9.22,1.71,0.31,1
1.63,6.12,1.69,0.30,1
1.60,33.84,1.67,0.29,1
1.58,82.33,1.65,0.26,1
1.58,146.68,1.64,0.26,1
1.60,159.74,1.63,0.25,1
1.72,142.10,1.65,0.25,1
1.83,122.61,1.68,0.25,1
1.84,70.63,1.71,0.25,1
1.90,45.30,1.75,0.25,1
1.84,71.63,1.77,0.25,1
2.00,30.06,1.82,0.25,1
2.00,59.55,1.85,0.25,1
1.95,57.52,1.87,0.25,1
1.80,82.37,1.86,0.24,1
1.63,111.60,1.81,0.24,1
1.64,100.61,1.78,0.24,1
1.93,111.94,1.81,0.24,1
1.99,117.89,1.84,0.24,1
2.63,220.61,2.00
1.74,40.79,1.95,0.37,2
1.72,39.32,1.90,0.37,2
1.77,39.02,1.88,0.37,2
1.81,40.64,1.86,0.37,2
1.73,42.31,1.84,0.37,2
1.73,69.86,1.81,0.35,2
```

Pessoa Caindo

- MAG (|a|, g): média 1,178, desvio 0,217, p95 1,660, máx 2,060

```
X,Y,Z,MAG,SMOOTH,STATE
0.11,-0.65,1.13,1.31,1.31,0
0.05,-0.66,1.04,1.23,1.29,0
0.09,-0.65,0.96,1.17,1.27,0
0.16,-0.57,0.95,1.12,1.24,0
0.14,-0.53,0.81,0.98,1.19,0
0.03,-0.49,0.74,0.89,1.13,0
-0.15,-0.43,0.69,0.83,1.07,0
-0.06,-0.50,0.72,0.88,1.03,0
-0.34,-0.33,0.98,1.09,1.04,0
-0.40,-0.23,0.92,1.03,1.04,0
-0.43,-0.18,0.90,1.02,1.04,0
-0.45,-0.11,0.95,1.05,1.04,0
-0.47,-0.05,0.96,1.08,1.04,0
-0.46,-0.01,0.97,1.09,1.04,0
-0.45,0.01,0.99,1.10,1.04,0
-0.45,0.03,1.00,1.11,1.04,0
-0.46,0.04,0.99,1.11,1.04,0
-0.47,0.06,0.99,1.12,1.04,0
-0.43,0.06,1.00,1.11,1.04,0
-0.39,0.04,1.01,1.11,1.04,0
-0.32,0.01,1.01,1.11,1.04,0
-0.27,-0.03,1.02,1.11,1.04,0
-0.21,-0.07,1.03,1.13,1.05,0
-0.11,-0.10,1.04,1.14,1.05,0
-0.04,-0.11,1.04,1.15,1.06,0
0.02,-0.10,1.04,1.15,1.07,0
0.09,-0.09,1.03,1.14,1.08,0
0.18,-0.07,1.03,1.16,1.10,0
0.27,-0.03,1.02,1.19,1.12,0
0.31,0.03,1.01,1.22,1.15,0
```