Dashboard Técnico com IA + Modbus

Com Flask, Modbus TCP, Groq (IA), geração de PDF e configuração dinâmica

Autor: Leandro T. Souza (projeto)
Gerado em: 11/08/2025 17:46:26

Sumário

1. Visão Geral

Este documento apresenta dois componentes que, juntos, formam uma solução de monitoramento e análise com IA:

1) servidor09_modbus.py — um servidor Modbus TCP de testes usando **pymodbus** para simular registradores.
2) dash10_groq377_modbus.py — um dashboard **Flask** responsivo que lê registradores via **pyModbusTCP**, plota gráficos, mantém histórico, gera PDF, e integra **IA** (Groq).

Você encontrará: capa, sumário, visão geral, dependências, comandos de instalação/execução para Linux e Windows, tecnologias usadas (com links) e os códigos completos.

2. Dependências

```
Dependências (mínimo):
- Python 3.9+
- pip

Bibliotecas Python:
- flask
- pyModbusTCP
- pymodbus   (apenas para o servidor de teste)
- requests
- fpdf2      (no dashboard, para gerar PDF; no tutorial usamos reportlab ou fpdf2)
- (opcional) reportlab

Comandos de instalação (Linux/macOS/Termux):
    python3 -m pip install --upgrade pip
    pip install flask pyModbusTCP pymodbus requests fpdf2 reportlab

Comandos de instalação (Windows PowerShell):
    py -m pip install --upgrade pip
    py -m pip install flask pyModbusTCP pymodbus requests fpdf2 reportlab
```

3. Como executar

```
Como executar — Passo a passo

1) Inicie o servidor Modbus de testes (porta 5020):
   Linux/Termux/macOS:
       python3 servidor09_modbus.py
   Windows (PowerShell):
       py servidor09_modbus.py

2) Em outro terminal, inicie o dashboard Flask (porta 5000):
   Linux/Termux/macOS:
       python3 dash10_groq377_modbus.py
   Windows (PowerShell):
       py dash10_groq377_modbus.py

3) Acesse pelo navegador:
   http://127.0.0.1:5000/        (página inicial)
   http://127.0.0.1:5000/grafico (gráfico, campos para Modelo e API Key)

4) Configure a IA (Groq) pela página /grafico:
   - Informe o **Modelo** (ex.: llama-3.1-8b-instant, llama-3.3-70b-versatile, deepseek-r1-distill-llama
   - Informe a **API Key** da Groq (formato gsk_...)
   - Clique **Salvar** e **Testar chave**.

5) Envie perguntas técnicas à IA, gere PDF da resposta e baixe o log.
```

4. Tecnologias usadas

Tecnologias usadas (referências):
- Flask (microframework web em Python) — https://flask.palletsprojects.com/
- pyModbusTCP (cliente Modbus TCP) — https://github.com/sourceperl/pyModbusTCP
- pymodbus (servidor/cliente Modbus) — https://github.com/pymodbus-dev/pymodbus
- Chart.js (gráficos no front-end) — https://www.chartjs.org/
- Groq (API compatível com OpenAI) — https://console.groq.com/docs
- Requests (HTTP client em Python) — https://requests.readthedocs.io/
- fpdf2 (geração de PDF em Python) — https://pyfpdf.github.io/fpdf2/

Código completo: servidor09_modbus.py

```
from pymodbus.server.sync import StartTcpServer
from pymodbus.datastore import ModbusSlaveContext, ModbusServerContext
from pymodbus.datastore.store import ModbusSequentialDataBlock
import random
import time
import threading

# Cria registradores com 100 posições
store = ModbusSlaveContext(
    hr=ModbusSequentialDataBlock(0, [0]*100)
)
context = ModbusServerContext(slaves=store, single=True)

def atualizar_registradores():
    while True:
        valores = [
            int(random.uniform(2100, 2400)),  # tensão
            int(random.uniform(950, 1100)),   # corrente
            int(random.uniform(90, 100)),     # FP
            int(random.uniform(20, 40)),      # temperatura
            int(random.uniform(80, 150)),     # vazão
            int(random.uniform(600, 1000)),   # nível
            int(random.uniform(690, 740)),    # pH
            int(random.uniform(400, 800))     # umidade
        ]
        for i, val in enumerate(valores):
            context[0x00].setValues(3, i, [val])
        print("■ Registradores atualizados:", valores)
        time.sleep(2)

# Roda o servidor em thread separada
threading.Thread(target=atualizar_registradores, daemon=True).start()

print("■ Servidor pymodbus rodando na porta 5020")
StartTcpServer(context, address=("0.0.0.0", 5020))
```

Código completo: dash10_groq377_modbus.py

```
# dashboard.py  (Flask + Modbus + IA via Groq + PDF + Config dinâmica)
from flask import Flask, render_template_string, jsonify, send_file, request
from pyModbusTCP.client import ModbusClient
from statistics import mean, stdev
from fpdf import FPDF  # pip install fpdf2
import time
import io
import os
import requests

app = Flask(__name__)

# -------------------- Estado --------------------
hist = []
ULTIMA_RESPOSTA_IA = ""

# -------------------- MODBUS --------------------
MODBUS_IP = "127.0.0.1"
MODBUS_PORT = 5020
client = ModbusClient(host=MODBUS_IP, port=MODBUS_PORT, auto_open=True)

def ler_modbus():
    regs = client.read_holding_registers(0, 8)
    if not regs:
        return {k: 0 for k in base}
    return {
        "tensao": round(regs[0] * 0.1, 1),
        "corrente": round(regs[1] * 0.01, 2),
        "fp": round(regs[2] * 0.01, 2),
        "temperatura": regs[3],
        "vasao": round(regs[4] * 0.1, 1),
        "nivel": round(regs[5] * 0.1, 1),
        "ph": round(regs[6] * 0.01, 2),
        "umidade": round(regs[7] * 0.1, 1)
    }

base = {
    "tensao": 220, "corrente": 10, "fp": 1.0, "temperatura": 25,
    "vasao": 10, "nivel": 80, "ph": 7, "umidade": 50
}
VAR_ORDER = ["tensao","corrente","fp","temperatura","vasao","nivel","ph","umidade"]

def resumo_estatistico(amostras):
    res = {}
    for k in VAR_ORDER:
        vals = [a[k] for a in amostras if k in a]
        if not vals:
            res[k] = {"min": None, "max": None, "media": None, "desvio": None}
        else:
            res[k] = {
                "min": min(vals),
                "max": max(vals),
                "media": round(mean(vals), 3),
                "desvio": round(stdev(vals), 3) if len(vals) > 1 else 0.0
            }
    return res

def series_tendencia(amostras, max_pontos=60):
    tail = amostras[-max_pontos:]
    series = {"tempo": [a.get("tempo","") for a in tail]}
    for k in VAR_ORDER:
        series[k] = [a.get(k, 0) for a in tail]
    return series

def csv_do_hist(amostras):
    linhas = ["tempo," + ",".join(VAR_ORDER)]
    for d in amostras:
        linhas.append(",".join(str(d.get(k,"")) for k in ["tempo", *VAR_ORDER]))
    return "\n".join(linhas)

# -------------------- IA via Groq (config dinâmica) --------------------
GROQ_CHAT = "https://api.groq.com/openai/v1/chat/completions"
GROQ_MODELS = "https://api.groq.com/openai/v1/models"

# Valores padrão (podem ser trocados pela UI /config)
CURRENT_API_KEY = "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"
CURRENT_MODEL = "llama-3.1-8b-instant"
def _groq_headers():
```

```
    return {"Authorization": f"Bearer {CURRENT_API_KEY}", "Content-Type": "application/json"}

def _construir_contexto_ia(pergunta: str):
    atual = ler_modbus()
    atual["tempo"] = time.strftime("%H:%M:%S")
    hist.append(atual)
    if len(hist) > 200:
        hist.pop(0)

    log_tail = hist[-200:]
    resumo = resumo_estatistico(log_tail) if log_tail else {}
    series = series_tendencia(log_tail, max_pontos=60) if log_tail else {"tempo": []}
    log_csv = csv_do_hist(log_tail)

    MAX_CSV = 12000
    if len(log_csv) > MAX_CSV:
        log_csv = log_csv[:MAX_CSV] + "\n...TRUNCADO..."

    return {
        "snapshot_atual": {k: atual[k] for k in ["tempo", *VAR_ORDER]},
        "base_referencia": base,
        "log_resumo": resumo,
        "series_tendencia": series,
        "log_csv": log_csv
    }

# -------------------- PDF --------------------
def _try_set_unicode_font(pdf: FPDF) -> bool:
    candidates = [
        "/system/fonts/Roboto-Regular.ttf",
        "/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf",
        "C:\\Windows\\Fonts\\arial.ttf"
    ]
    for path in candidates:
        if os.path.exists(path):
            try:
                pdf.add_font("UX", "", path, uni=True)
                pdf.set_font("UX", size=11)
                return True
            except Exception:
                pass
    return False

def _sanitize_latin1(texto: str) -> str:
    return (texto or "").encode("latin-1", "ignore").decode("latin-1")

def gerar_pdf_resposta(texto: str) -> bytes:
    try:
        pdf = FPDF()
        pdf.set_auto_page_break(auto=True, margin=15)
        pdf.add_page()
        pdf.set_font("Helvetica", "B", 14)
        pdf.cell(0, 10, "Resposta da IA", ln=True)

        if not _try_set_unicode_font(pdf):
            pdf.set_font("Helvetica", size=11)
            texto = _sanitize_latin1(texto)

        pdf.multi_cell(0, 6, texto if texto else "Sem resposta.")
        pdf.ln(4)
        if pdf.font_family != "UX":
            pdf.set_font("Helvetica", size=9)
        else:
            pdf.set_font("UX", size=9)
        pdf.multi_cell(0, 5, f"Gerado em {time.strftime('%d/%m/%Y %H:%M:%S')}")
        out = pdf.output(dest="S")
        return out.encode("latin-1", "ignore") if isinstance(out, str) else out
    except Exception as e:
        pdf = FPDF()
        pdf.add_page()
        pdf.set_font("Helvetica", size=12)
        pdf.multi_cell(0, 6, _sanitize_latin1(f"Falha ao gerar PDF: {e}"))
        out = pdf.output(dest="S")
        return out.encode("latin-1", "ignore") if isinstance(out, str) else out

# -------------------- Rotas de dados --------------------
@app.route('/dados')
def dados():
    valores = ler_modbus()
    valores["tempo"] = time.strftime("%H:%M:%S")
    hist.append(valores)
```

```
    if len(hist) > 200:
        hist.pop(0)
    alarme = any(valores[k] < base[k]*0.5 for k in base)
    return jsonify(valores | {"alarme": alarme})

@app.route('/historico')
def historico():
    return jsonify(hist)

@app.route('/download-log')
def download_log():
    txt = csv_do_hist(hist)
    return send_file(io.BytesIO(txt.encode()), download_name="log.txt", as_attachment=True)

# -------------------- Rotas de IA --------------------
@app.route("/perguntar", methods=["POST"])
def perguntar():
    global ULTIMA_RESPOSTA_IA
    pergunta = (request.json or {}).get("pergunta", "").strip()
    if not pergunta:
        return jsonify({"resposta": "■ Pergunta vazia."})
    try:
        contexto = _construir_contexto_ia(pergunta)
        payload = {
            "model": CURRENT_MODEL,
            "messages": [
                {"role": "system", "content": "Você é uma IA técnica de automação industrial. Seja preci
                {"role": "user", "content": f"CONTEXTO (JSON):\n{contexto}"},
                {"role": "user", "content": f"PERGUNTA: {pergunta}"}
            ],
            "temperature": 0.2,
            "max_tokens": 700
        }
        r = requests.post(GROQ_CHAT, headers=_groq_headers(), json=payload, timeout=90)
        r.raise_for_status()
        j = r.json()
        resposta = j.get("choices", [{}])[0].get("message", {}).get("content", "").strip()
        if not resposta:
            resposta = "■ Não foi possível extrair a resposta da IA."
        ULTIMA_RESPOSTA_IA = resposta
        return jsonify({"resposta": resposta})
    except requests.Timeout:
        return jsonify({"resposta": "■ Timeout (90s) ao consultar a IA."})
    except Exception as e:
        return jsonify({"resposta": f"■ Erro ao consultar a IA: {e}"})

@app.route("/testar-chave")
def testar_chave():
    try:
        r = requests.get(GROQ_MODELS, headers={"Authorization": f"Bearer {CURRENT_API_KEY}"}, timeout=15
        if r.status_code == 200:
            return jsonify({"ok": True, "mensagem": "■ Chave API válida!"})
        return jsonify({"ok": False, "mensagem": f"■ Falha ({r.status_code}): {r.text[:200]}"})
    except requests.Timeout:
        return jsonify({"ok": False, "mensagem": "■ Timeout ao validar a chave (15s)."})
    except Exception as e:
        return jsonify({"ok": False, "mensagem": f"■ Erro: {e}"})

# -------------------- Config dinâmica (API key + modelo) --------------------
@app.route("/config", methods=["GET", "POST"])
def config():
    global CURRENT_API_KEY, CURRENT_MODEL
    if request.method == "POST":
        j = request.get_json(force=True, silent=True) or {}
        api_key = (j.get("api_key") or "").strip()
        model = (j.get("model") or "").strip()
        if api_key:
            CURRENT_API_KEY = api_key
        if model:
            CURRENT_MODEL = model
        return jsonify({"ok": True, "msg": "Configuração atualizada com sucesso."})
    # GET: retorna sem expor a chave inteira
    masked = (CURRENT_API_KEY[:6] + "..." + CURRENT_API_KEY[-4:]) if CURRENT_API_KEY else None
    return jsonify({"model": CURRENT_MODEL, "api_key_masked": masked})

# -------------------- PDF da resposta --------------------
@app.route('/ia-pdf-download', methods=['GET'])
def ia_pdf_download():
    texto = ULTIMA_RESPOSTA_IA or "Sem resposta disponível. Faça uma pergunta à IA primeiro."
    binario = gerar_pdf_resposta(texto)
    return send_file(io.BytesIO(binario),
```

```
                    mimetype="application/pdf",
                    as_attachment=True,
                    download_name="resposta_IA.pdf")

# ---- Endpoints de debug opcionais ----
@app.route('/__healthz')
def __healthz():
    return jsonify({"ok": True, "routes": len(list(app.url_map.iter_rules()))})

@app.route('/__routes')
def __routes():
    rotas = []
    for r in app.url_map.iter_rules():
        rotas.append({
            "rule": str(r),
            "methods": sorted([m for m in r.methods if m not in ("HEAD","OPTIONS")])
        })
    return jsonify(rotas)

# -------------------- HTML (mantendo estética e responsividade) --------------------
HTML_GAUGES = """<html>
<head>
  <meta charset="utf-8">
  <title>Dashboard - Gauges</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/chartjs-plugin-doughnutlabel@1.0.3"></script>
  <style>
    body { max-width: 100vw; overflow-x: hidden; }
    body { font-family: sans-serif; margin:0; padding:10px; background: #f8f8f8; }
    .btn { padding: 10px 16px; font-size: 16px; background: #2196f3; color: white; border: none; border-
    #painel { display: flex; flex-wrap: wrap; justify-content: center; gap: 12px; }
    .gauge-box { width: 45%; max-width: 160px; background: #fff; padding: 8px; border-radius: 10px; box-
    @media (max-width: 600px) { .gauge-box { width: 46%; } }
  </style>
</head>
<body>
<h3 style="text-align:center">■ Dashboard Técnico</h3>
<div id="painel"></div>
<a href="/grafico"><button class="btn">■ Ver gráfico de tendência</button></a>
<script>
const vars = [
  {id:"tensao", label:"Tensão (V)", cor:"gold", max:300},
  {id:"corrente", label:"Corrente (A)", cor:"dodgerblue", max:20},
  {id:"fp", label:"Fator Potência", cor:"gray", max:1},
  {id:"temperatura", label:"Temperatura (°C)", cor:"red", max:100},
  {id:"vasao", label:"Vazão (L/min)", cor:"limegreen", max:30},
  {id:"nivel", label:"Nível (%)", cor:"blue", max:100},
  {id:"ph", label:"pH", cor:"purple", max:14},
  {id:"umidade", label:"Umidade (%)", cor:"teal", max:100}
];
const charts = {};
function criarGauges(){
  const painel = document.getElementById("painel");
  vars.forEach(v => {
    const div = document.createElement("div");
    div.className = "gauge-box";
    div.innerHTML = `<canvas id="${v.id}" width="120" height="120"></canvas><div id="valor_${v.id}" styl
    painel.appendChild(div);
    const ctx = document.getElementById(v.id).getContext("2d");
    charts[v.id] = new Chart(ctx, {
      type:"doughnut",
      data:{datasets:[{data:[0,v.max], backgroundColor:[v.cor,"#eee"]}]},
      options:{cutout:"70%", plugins:{legend:{display:false}, doughnutlabel:{labels:[{text:"0", font:{si
    });
  });
}
function atualizar(){
  fetch("/dados").then(r=>r.json()).then(d=>{
    vars.forEach(v=>{
      charts[v.id].data.datasets[0].data = [d[v.id], v.max - d[v.id]];
      charts[v.id].options.plugins.doughnutlabel.labels[0].text = d[v.id];
      charts[v.id].update();
      document.getElementById("valor_"+v.id).innerText = `${v.label}: ${d[v.id]}`;
    });
  });
}
criarGauges();
setInterval(atualizar, 2000);
</script>
</body></html>
```

```
"""

HTML_GRAFICO = """<html>
<head>
  <meta charset="utf-8">
  <title>Gráfico de Tendência</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <style>
    html, body { height: 100%; }
    body { max-width: 100vw; overflow-x: hidden; margin:0; padding:10px; font-family: sans-serif; backgr
    .btn { margin: 6px; padding: 8px 12px; background: #4caf50; color: white; border: none; border-radiu
    input, select { padding: 6px; margin: 4px 0; width: 90%; max-width: 520px; }
    label { display:block; margin-top:8px; font-size:14px; color:#333; }
    canvas { display:block; width: 100% !important; height: auto !important; }
    #grafico { width: 100% !important; height: 60vh !important; max-height: 75vh; }
    @media (max-width: 480px) { #grafico { height: 62vh !important; } }
    .card { background:#fff; padding:10px; border-radius:8px; box-shadow:0 1px 4px rgba(0,0,0,0.1); marg
  </style>
</head>
<body>
<h3>■ Gráfico de Tendência</h3>
<a href="/"><button class="btn">■■ Voltar aos gauges</button></a>

<div class="card">
  <h4>■■ Configuração de IA (Groq)</h4>
  <label>Modelo (ex.: llama-3.1-8b-instant, llama-3.3-70b-versatile, deepseek-r1-distill-llama-70b)</lab
  <input id="modelo" placeholder="llama-3.1-8b-instant">
  <label>API Key (Groq)</label>
  <input id="apikey" placeholder="gsk_..." type="password">
  <button class="btn" onclick="salvarConfig()">■ Salvar</button>
  <button class="btn" onclick="testarChave()">■ Testar chave</button>
  <p id="cfgstatus" style="font-size:14px;color:#444"></p>
</div>

<div style="width: 100%; max-width: 100%; overflow-x: hidden;" class="card">
  <canvas id="grafico"></canvas>
  <br>
  <input id="pergunta" placeholder="Pergunta técnica...">
  <button onclick="enviar()" class="btn">■ Perguntar à IA</button>
  <button onclick="baixarPDF()" class="btn">■ Baixar resposta (PDF)</button>
  <p id="resposta" style="background:#eee; padding:10px; border-radius:6px;"></p>
</div>

<div class="card">
  <button class="btn" onclick="baixarLog()">■ Baixar log.txt</button>
  <button class="btn" onclick="capturarGrafico()">■■ Capturar gráfico</button>
  <button class="btn" onclick="compartilhar()">■ Compartilhar</button>
</div>

<script>
const ctx = document.getElementById("grafico").getContext("2d");
const vars = [
  {id:"tensao", label:"Tensão (V)", cor:"gold"},
  {id:"corrente", label:"Corrente (A)", cor:"dodgerblue"},
  {id:"fp", label:"Fator Potência", cor:"gray"},
  {id:"temperatura", label:"Temperatura (°C)", cor:"red"},
  {id:"vasao", label:"Vazão (L/min)", cor:"limegreen"},
  {id:"nivel", label:"Nível (%)", cor:"blue"},
  {id:"ph", label:"pH", cor:"purple"},
  {id:"umidade", label:"Umidade (%)", cor:"teal"}
];

// --- Grafico ---
const graf = new Chart(ctx, {
  type: "line",
  data: { labels: [], datasets: vars.map(v => ({ label: v.label, data: [], borderColor: v.cor, fill: fal
  options: {
    responsive: true,
    maintainAspectRatio: false,
    animation: false,
    scales: { x: { title: { display: true, text: "Tempo"}}, y: { beginAtZero: true } }
  }
});

function atualizar(){
  fetch("/dados").then(r => r.json()).then(d => {
    graf.data.labels.push(d.tempo);
    if (graf.data.labels.length > 100) graf.data.labels.shift();
    vars.forEach((v, i) => {
      graf.data.datasets[i].data.push(d[v.id]);
```

```
      if (graf.data.datasets[i].data.length > 100) graf.data.datasets[i].data.shift();
    });
    graf.update();
  });
}
setInterval(atualizar, 2000);

// --- Config dinâmica ---
function carregarConfig(){
  // Prefill do servidor
  fetch("/config").then(r => r.json()).then(c => {
    const m = localStorage.getItem("modelo") || c.model || "llama-3.1-8b-instant";
    const k = localStorage.getItem("apikey") || "";
    document.getElementById("modelo").value = m;
    document.getElementById("apikey").value = k;
    document.getElementById("cfgstatus").innerText = "Modelo atual (servidor): " + (c.model || "");
  });
}
function salvarConfig(){
  const model = document.getElementById("modelo").value.trim();
  const api_key = document.getElementById("apikey").value.trim();
  localStorage.setItem("modelo", model);
  if (api_key) localStorage.setItem("apikey", api_key);
  fetch("/config", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ model, api_key })
  }).then(r => r.json()).then(d => {
    document.getElementById("cfgstatus").innerText = d.ok ? "■ Config salva no servidor." : "■ Falha ao
  }).catch(_ => document.getElementById("cfgstatus").innerText = "■ Erro ao salvar.");
}
function testarChave(){
  const maybeNewKey = document.getElementById("apikey").value.trim();
  if (maybeNewKey) {
    // salva antes de testar
    fetch("/config", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ api_key: maybeNewKey })
    }).then(_ => {
      fetch("/testar-chave").then(r => r.json()).then(d => alert(d.mensagem));
    });
  } else {
    fetch("/testar-chave").then(r => r.json()).then(d => alert(d.mensagem));
  }
}

// --- Utilitários ---
function baixarLog(){ window.location = "/download-log"; }
function capturarGrafico(){
  const link = document.createElement("a");
  link.download = "grafico.png";
  link.href = document.getElementById("grafico").toDataURL("image/png");
  link.click();
}
function compartilhar(){
  const texto = "Veja o gráfico de tendência em tempo real!";
  if (navigator.share) navigator.share({ title: "Dashboard Modbus", text: texto }).catch(_=>{});
  else alert("Navegador não suporta compartilhamento.");
}

// --- IA ---
function enviar(){
  const p = document.getElementById("pergunta").value;
  document.getElementById("resposta").innerText = "■ consultando IA...";
  fetch("/perguntar", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ pergunta: p })
  }).then(r => r.json()).then(d => {
    document.getElementById("resposta").innerText = d.resposta;
  }).catch(_ => {
    document.getElementById("resposta").innerText = "■ Erro de rede.";
  });
}
function baixarPDF(){
  window.location = "/ia-pdf-download?t=" + Date.now();
}

// Inicializa
carregarConfig();
```

```
</script>
</body></html>
"""

# -------------------- Rotas de página --------------------
@app.route('/')
def gauges():
    return render_template_string(HTML_GAUGES)

@app.route('/grafico')
def grafico():
    return render_template_string(HTML_GRAFICO)

# -------------------- MAIN --------------------
if __name__ == '__main__':
    # Dependências: pip install flask pyModbusTCP requests fpdf2
    app.run(host="0.0.0.0", port=5000)
```