```python
import os
from flask import Flask
import requests
from datetime import datetime
import dash
from dash import dcc, html
from dash.dependencies import Input, Output, State
import json

# ----------------------------
# Configuração
# ----------------------------
server = Flask(__name__)
ESP32_IP = os.getenv("ESP32_IP", "10.244.60.22")  # <<< AJUSTE SEU IP AQUI
CLASSIFY_URL = f"http://{ESP32_IP}/classificar"
#http://10.244.60.22/classificar
# Estado inferido dos LEDs (baseado nas respostas do ESP32)
led_state = {
    "pos1": False,
    "pos2": False,
    "pos3": False,
    "pos4": False,
    "pos5": False,
    "pos6": False,
    "alerta": False
}
last_response = ""
last_qr = ""
last_update = "Nunca"
connection_status = "Aguardando envio"

# ----------------------------
# Função para enviar QR e atualizar estado
# ----------------------------
def classify_and_update(qr_code):
    global led_state, last_response, connection_status, last_update, last_qr
    last_qr = qr_code
    try:
        resp = requests.post(CLASSIFY_URL, json={"qr": qr_code}, timeout=3)
        if resp.status_code == 200:
            try:
                data = resp.json()
                last_response = json.dumps(data, indent=2, ensure_ascii=False)
                connection_status = "✅ Sucesso"
                # Resetar todos
                for k in led_state:
                    led_state[k] = False
                # Atualizar com base na resposta
                if data.get("status") == "ok":
                    pos = data.get("posicao", 0)
                    if 1 <= pos <= 6:
                        led_state[f"pos{pos}"] = True
                elif "invalid" in data.get("status", ""):
                    led_state["alerta"] = True
            except:
                last_response = "Resposta inválida do ESP32"
                led_state["alerta"] = True
                connection_status = "❌ Resposta inválida"
        else:
            last_response = f"Erro HTTP: {resp.status_code}"
            led_state["alerta"] = True
            connection_status = "❌ Erro de requisição"
    except Exception as e:
        last_response = str(e)
        led_state["alerta"] = True
        connection_status = "❌ Falha de conexão"
    last_update = datetime.now().strftime("%H:%M:%S")

# ----------------------------
# Estilo de LED
# ----------------------------
def led_indicator(is_on, label):
    color = "#4CAF50" if is_on else "#9E9E9E"
    return html.Div([
        html.Div(style={
            "width": "40px",
            "height": "40px",
            "borderRadius": "50%",
            "backgroundColor": color,
            "boxShadow": "0 0 8px rgba(0,0,0,0.3)",
            "margin": "auto"
        }),
        html.Div(label, style={"textAlign": "center", "marginTop": "5px", "fontSize": "14px"})
    ], style={"textAlign": "center", "margin": "10px", "width": "80px"})

# ----------------------------
# Layout do Dashboard
# ----------------------------
app = dash.Dash(__name__, server=server, url_base_pathname="/")
app.layout = html.Div([
    html.H1("🚦 Dashboard de Classificação QR Code", style={"textAlign": "center", "color": "#2c3e50"}),

    html.Div(id="status-banner", style={"textAlign": "center", "margin": "10px", "fontWeight": "bold"}),

    html.Div([
        dcc.Input(
            id="input-qr",
            type="text",
            placeholder="Ex: CX12345",
            style={"width": "300px", "padding": "10px", "fontSize": "16px", "marginRight": "10px"}
        ),
        html.Button("Enviar para ESP32", id="btn-send", n_clicks=0,
                    style={"padding": "10px 20px", "fontSize": "16px", "backgroundColor": "#3498db", "color": "white", "border": "none", "borderRadius": "5px"}),
    ], style={"textAlign": "center", "margin": "20px"}),

    html.Div(id="response-output", style={"textAlign": "center", "margin": "20px", "whiteSpace": "pre-line", "fontFamily": "monospace"}),

    html.H2("Estado dos LEDs (Inferido)", style={"textAlign": "center", "margin": "30px 0"}),

    html.Div([
        led_indicator(led_state["pos1"], "Posição 1"),
        led_indicator(led_state["pos2"], "Posição 2"),
        led_indicator(led_state["pos3"], "Posição 3"),
        led_indicator(led_state["pos4"], "Posição 4"),
        led_indicator(led_state["pos5"], "Posição 5"),
        led_indicator(led_state["pos6"], "Posição 6"),
        led_indicator(led_state["alerta"], "⚠ Alerta"),
    ], style={"display": "flex", "justifyContent": "center", "flexWrap": "wrap"})
], style={"fontFamily": "Arial, sans-serif", "maxWidth": "1000px", "margin": "0 auto", "padding": "20px"})

# ----------------------------
# Callback para envio
# ----------------------------
@app.callback(
    [Output("response-output", "children"),
     Output("status-banner", "children")],
    Input("btn-send", "n_clicks"),
    State("input-qr", "value"),
    prevent_initial_call=True
)
def handle_qr_send(n_clicks, qr_value):
    if not qr_value or not qr_value.strip():
        return "⚠ Por favor, insira um QR Code.", "Aguardando envio"
    classify_and_update(qr_value.strip())
    banner = f"{connection_status} | Última atualização: {last_update}"
    return f"QR Enviado: {last_qr}\nResposta do ESP32:\n{last_response}", banner

# ----------------------------
# Execução
# ----------------------------
if __name__ == "__main__":
    #app.run(debug=True, host="0.0.0.0", port=8050)
    #app.run(host='0.0.0.0', port=5000, debug=False, threaded=True)
    app.run(host="0.0.0.0", debug=False, threaded=True, port=5000)
```