

# Bancada com Embarcados

**Controle IoT com Inteligência  
Artificial em Aplicações de  
Produtividade.**

**Apresentador:** Guilherme Fernandes de Oliveira

# Sobre mim

- São José dos Campos - SP
- Técnico em Automação pela ETEC
- Graduando em Engenharia Eletrônica UNIFEI
- 2 anos com desenvolvimento Web e Mobile
- Praticar esportes
- Ler



# Introdução

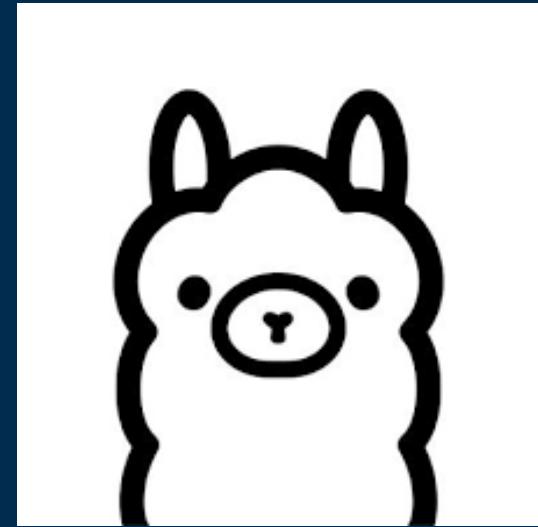
- Rotina intensa na bancada: estudos, soldagem, projetos e ambientes pouco otimizados
- Franzininho WiFi Lab01 com servidor Python e Ollama em Docker
- Uso de SLM + RAG para interpretar comandos e consultar documentação técnica em tempo real

# Objetivos

- Como está o ambiente para estudar?
- Ligue o Led Vermelho
- Inicie um pomodoro de 3 minutos
- Qual pino está o DHT11 na Franzininho?

# Tecnologias

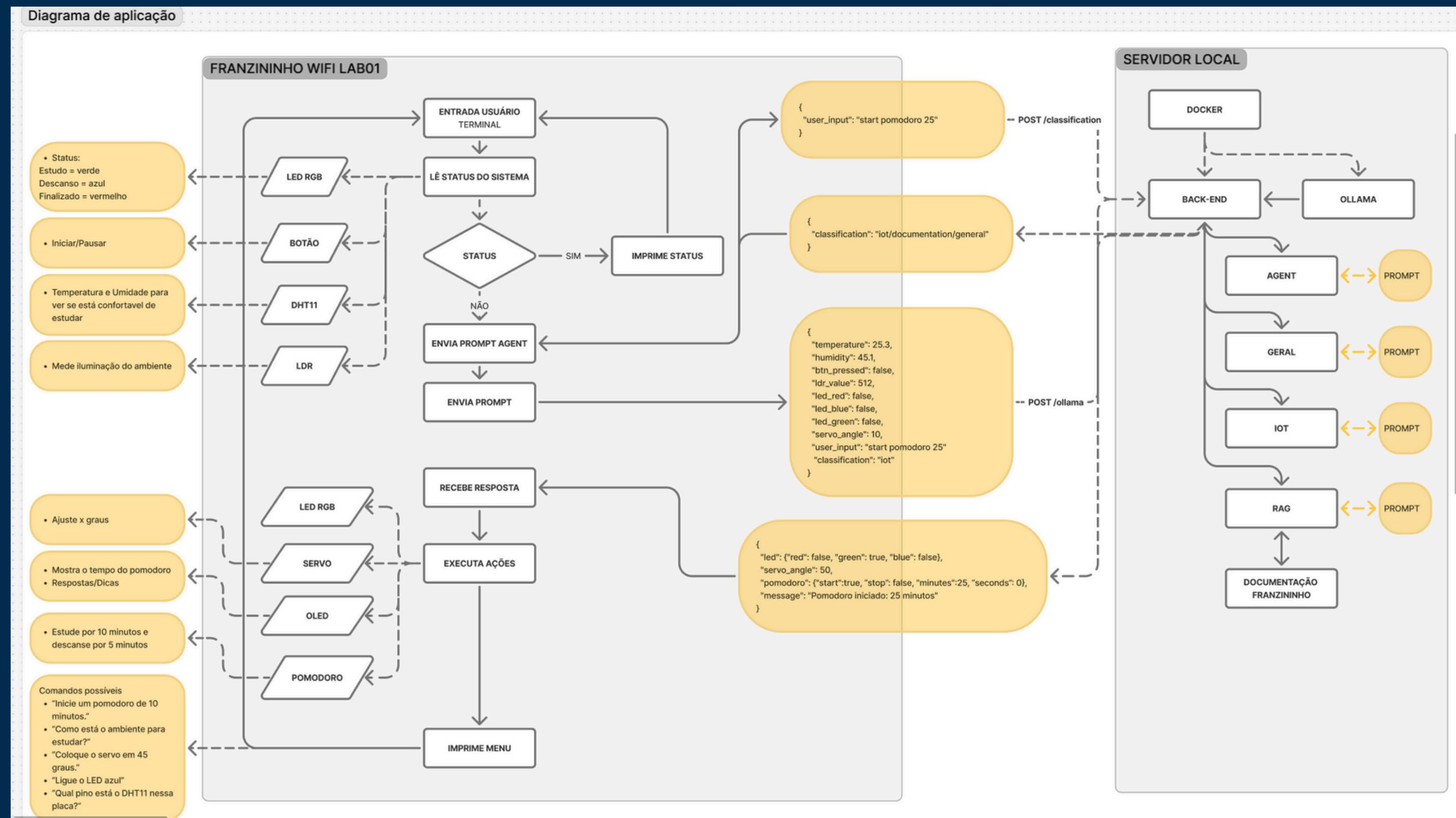
- Python
- Flask
- Ollama
- Gemma3
- Docker



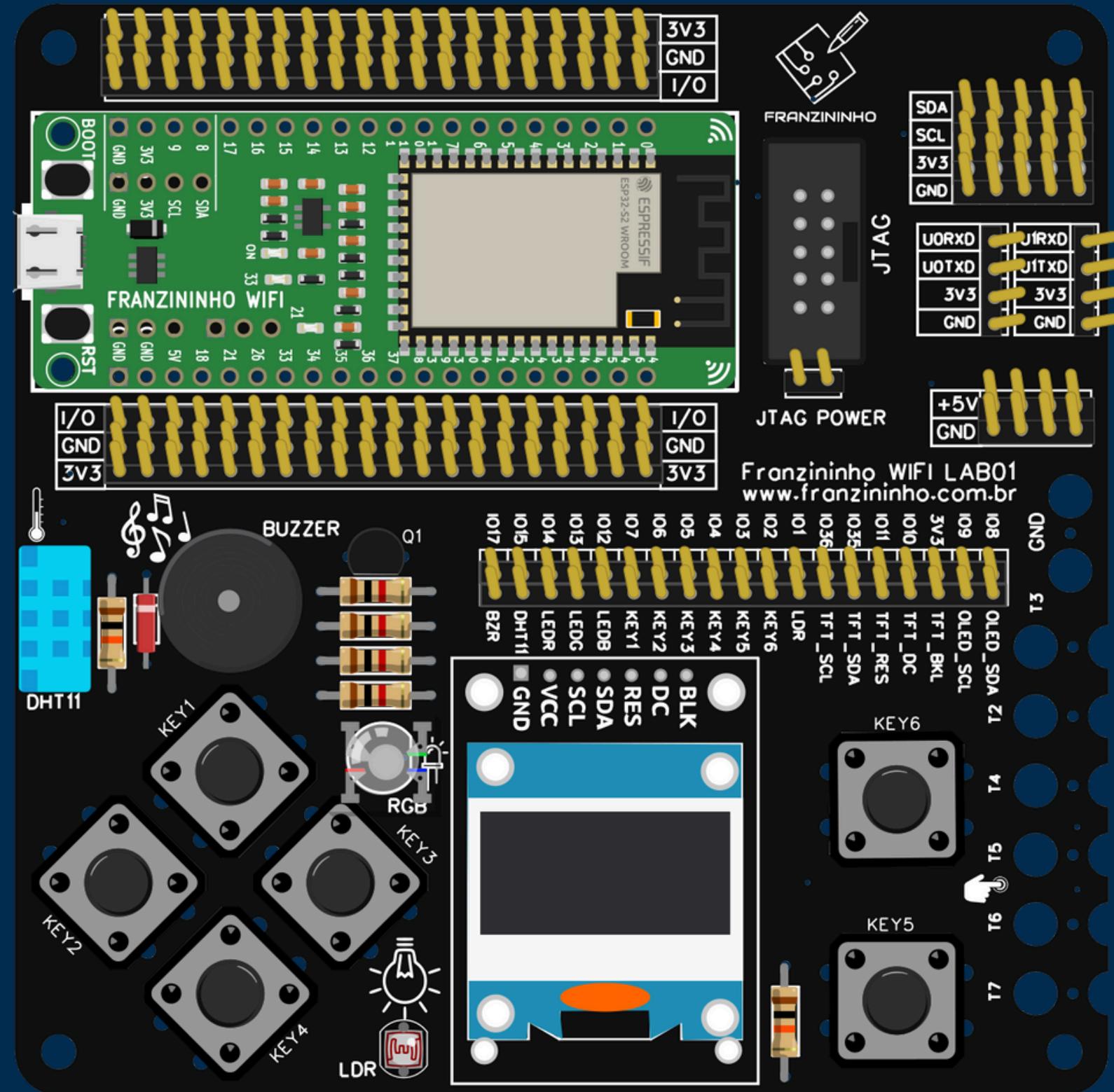
Gemma 3



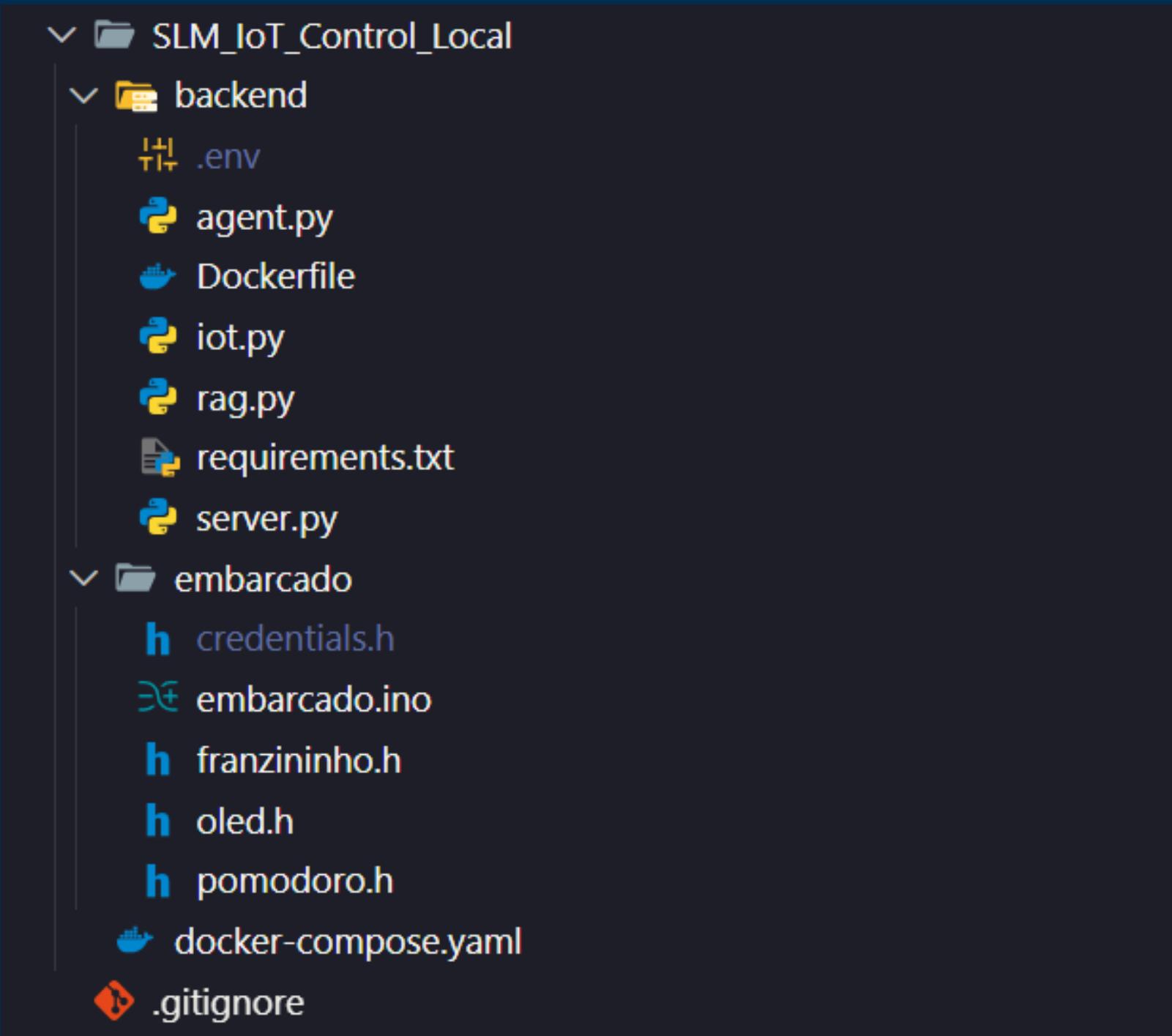
# Diagrama da aplicação



# Circuito



# Estrutura do Projeto



# Backend

```
@app.route("/ollama", methods=["POST"])
def ollama():
    body = rq.get_json()

    classification = body["classification"]

    if classification == "iot":
        message,(red, blue, green), servo_angle, (pomodoro_start, pomodoro_stop, pomodoro_minutes), response = useIOT.query(body)
        return jsonify({
            "message": message,
            "red_led": red,
            "blue_led": blue,
            "green_led": green,
            "servo_angle": servo_angle,
            "pomodoro": {
                "start": pomodoro_start,
                "stop": pomodoro_stop,
                "minutes": pomodoro_minutes,
            },
            "response": response
        })

    elif classification == "documentation":
        message = useRAG.query(body["user_input"])
        return jsonify({
            "message": message
        })

    else:
        message = useAGENT.ask_ollama(body["user_input"])
        return jsonify({
            "message": message
        })
```

# Backend

```
41  class RAG:
42      def __init__(self):
43          self.chunk_size = chunk_size
44          self.chunk_overlap = chunk_overlap
45          self.vectorstore = None
46          self.retriever = None
47
48          # self.llm = ChatOllama(model=self.model, temperature=0)
49
50          self.preload_models()
51          self.create_vectorstore(self.urls, self.pdfs, self.text, recreate=False)
52          self.load_vectorstore()
53
54          # -----
55          # Custom embedding class that uses Ollama directly and implements caching
56          # -----
57
58      > class OptimizedOllamaEmbeddings:
59
60          # -----
61          # Model warm-up
62          # -----
63
64          > def preload_models(self):
65
66              # -----
67              # Vectorstore creation
68              # -----
69
70          > def create_vectorstore(self, urls=None, pdfs=None, text=None, recreate=False):
71
72              # -----
73              # Load existing vectorstore
74              # -----
75
76          > def load_vectorstore(self):
77
78              # -----
79              # Query RAG
80              # -----
81
82          > def query(self, question):...
```

useRAG = RAG(  
 persist\_dir="chroma\_db",  
 model=MODEL,  
 urls = [  
 "https://raw.githubusercontent.com/Franzininho/docs-franzininho-site/main/docs/FranzininhoWiLAB01/franzininho-wifi-lab01.md",  
 "https://docs.franzininho.com.br/docs/franzininho-wifi/franzininho-wifi/"  
 ],  
 pdfs= [],  
 text=[  
])

# Backend

```
class IOT:
    def __init__(
        self,
        model,
        ollama_host
    ):
        self.model = model,
        self.ollama_host = ollama_host
        self.session = requests.Session()
        self.session.headers.update({"Connection": "keep-alive"})

    def create_interactive_prompt(self, temp, hum, button_state, LedRed, LedBlue, LedGreen):
        pass

    def slm_inference(self, PROMPT):
        response = self.session.post(
            url=f"{self.ollama_host}/api/generate",
            json={
                "model": "gemma3",
                "prompt": PROMPT,
                "stream": False,
                "format": "json",
            }
        )
        print(response.json().get("response", {}))
        return response.json().get("response", {})

    def parse_interactive_response(self, response_text):
        pass

    def query(self, body):
```

You are an IoT SLM controller. Always respond ONLY with valid JSON.

SYSTEM STATUS:

- temperature: {temp:.1f}
- humidity: {hum:.1f}
- button\_pressed: {str(button\_state).lower()}
- leds: red={str(LedRed).lower()}, blue={str(LedBlue).lower()}, green={str(LedGreen).lower()}
- ldr\_value: {ldr\_value}
- servo\_angle: {servo\_angle}

RULES:

1. Output ONLY this JSON structure:  
{{  
 "message": "",  
 "leds": {{  
 "red\_led": bool,  
 "green\_led": bool,  
 "blue\_led": bool  
 }},  
 "servo\_angle": int,  
 "pomodoro": {[  
 "start": bool,  
 "stop": bool,  
 "minutes": int  
 ]}  
}}
2. Do NOT add any text outside the JSON.
3. If the user asks for information, keep all hardware states unchanged.
4. If the user requests an action, update the states.
5. Only ONE LED may be ON unless the user explicitly requests multiple.
6. If the user asks about the environment, evaluate study productivity based on SYSTEM STATUS.
7. Keep "message" short and clear.

USER INPUT: "{user\_input}"

# Embarcado

```
void loop() {
    if (Serial.available()) {
        // Get current system status
        struct LedStatus ledsts = readLedStatus();
        struct SensorData data = readSensors();
        // Check if sensor data is valid, if not, return
        if (!data.success) { ...
        // Handle status command
        if (input.equalsIgnoreCase("status")) {
            printSystemStatus(data.temperature, data.humidity, data.buttonPressed, ledsts.red, ledsts.blue, ledsts.green, data.ldrValue, servoAngle);
            return;
        }
        // Send data to the LLM and get the response
        struct responseLLM response = sendToLlm(data.temperature, data.humidity, data.buttonPressed, ledsts.red, ledsts.blue, ledsts.green, data.ldrValue, servoAngle, input);
        // Get SLM response
        Serial.println("\nAssistant: [Thinking...]");
        // Display assistant's message
        Serial.println("Assistant: " + response.message)
        // Mostra no display (resposta IA)
        showMessage("Assistant: " + response.message);
        if (!response.success) {
            return;
        }
        // Control LEDs based on response
        setLeds(response.leds.red, response.leds.blue, response.leds.green);
        // Ajusta servo
        servoAngle = response.servoAngle;
        // myServo.write(servoAngle);
        // Configura o pomodoro
        if (response.pomodoro.start) {
            int minutes = response.pomodoro.minutes;
            startPomodoro(minutes);
        } else if (response.pomodoro.stop) {
            stopPomodoro();
        }
        printMenu();
    }
}
```

# Docker

 Dockerfile  SLM\_IoT\_Control LocalBackend Dockerfile

```
1 FROM python:3.11-slim
2 # define diretório de trabalho
3 WORKDIR /usr/src/app
4 # copia dependências
5 COPY requirements.txt .
6 # instala dependências
7 RUN pip install --no-cache-dir -r requirements.txt
8 # permite prints no container
9 ENV PYTHONUNBUFFERED=1
10 # copia o projeto
11 COPY . .
12 # expõe a porta do flask
13 EXPOSE 5000
14 # comando para executar
15 CMD ["python", "server.py"]
```

```
docker-compose.yml SLM_IoT_Control_Local\docker-compose.yaml [ ] services
  docker-compose.yml - The Compose specification establishes a standard
  services:
    ollama:
      image: ollama/ollama:latest
      container_name: ollama
      ports:
        - "11434:11434"
      volumes:
        - ollama_data:/root/.ollama
      entrypoint: >
        bash -c "
          ollama serve &
          sleep 3 &&
          if ! ollama list | grep -q 'gemma3'; then
            echo 'Baixando modelo gemma3...';
            ollama pull gemma3;
            ollama pull nomic-embed-text;
          else
            echo 'Modelo gemma3 já está instalado.';
          fi &&
          wait
        "
    backend:
      build:
        context: ./backend
        dockerfile: Dockerfile
      container_name: backend_python
      ports:
        - "5000:5000"
      env_file:
        - ./backend/.env
      depends_on:
        - ollama
      volumes:
        ollama_data:
```

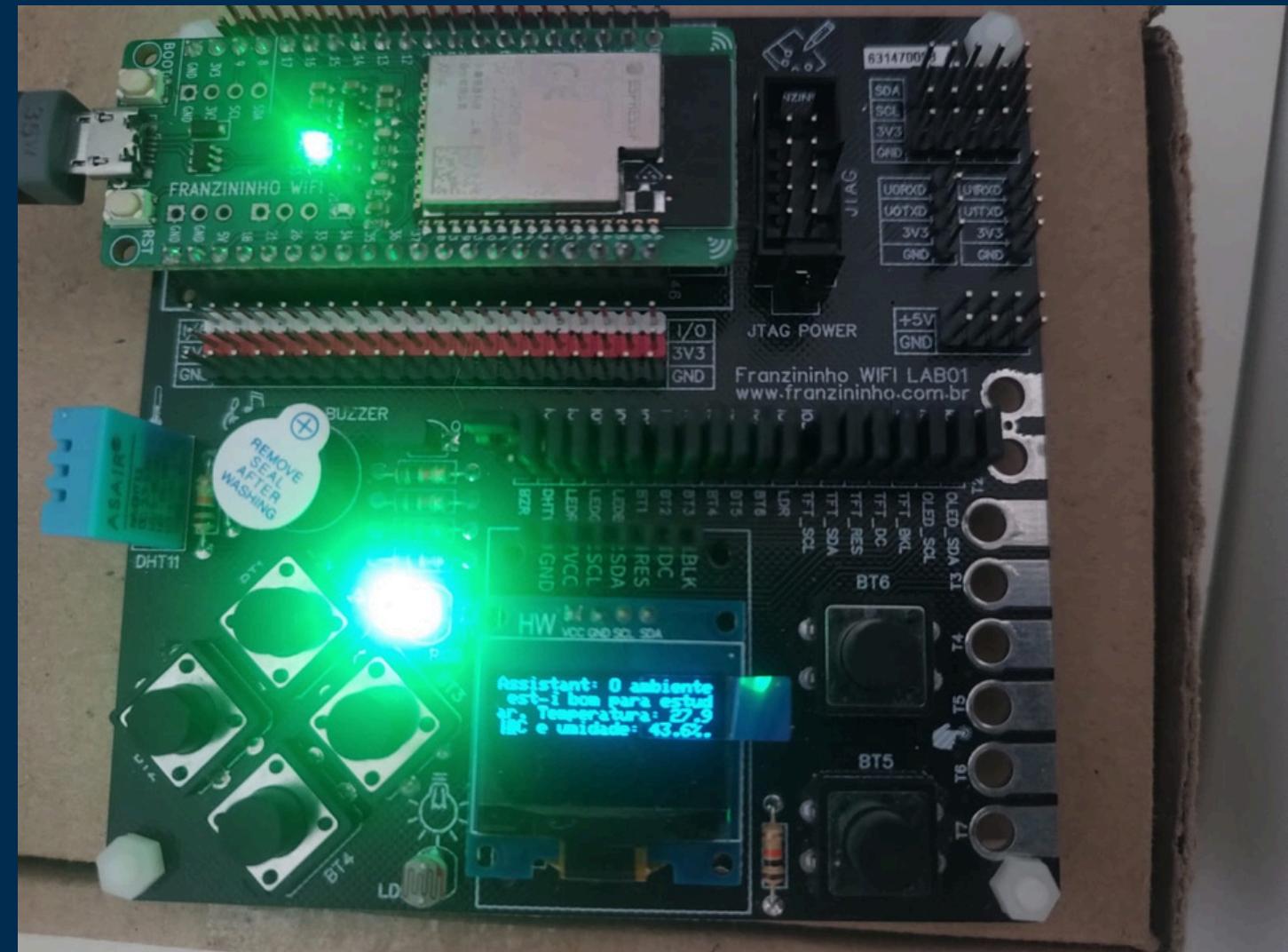
# “Como está o ambiente para estudar?”

```
=====
Controle IoT com SLM
Modelo usado: gemma3
=====

Comandos que você pode tentar:
- Como está o ambiente para estudar?
- Coloque o servo em 45 graus
- Ligue o LED azul
- Inicie um pomodoro de 10 minutos
- Qual pino está o DHT11 nessa placa?
- Qual é a temperatura atual?
- Desligue todos os LEDs
- Vai chover com base nas condições atuais?
- Digite 'status' para ver o status do sistema
=====

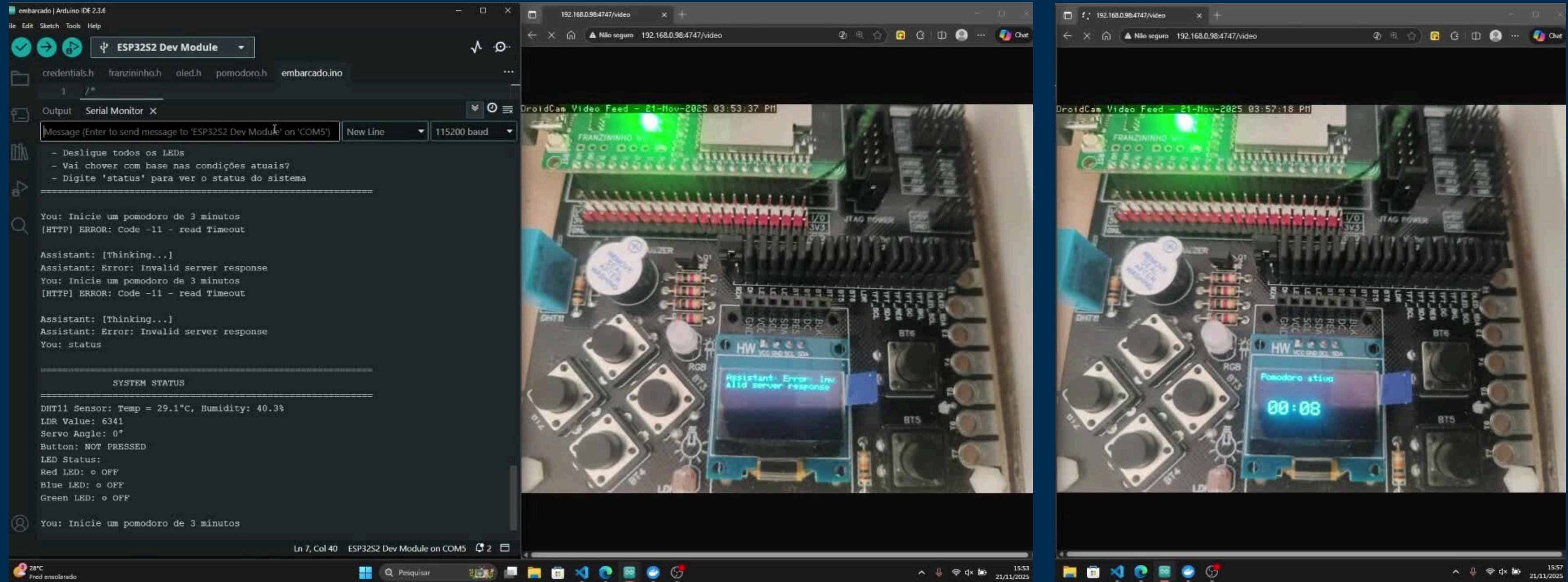
You: Como está o ambiente para estudar?

Assistant: [Thinking...]
Assistant: O ambiente está bom para estudar. Temperatura: 27.9°C e umidade: 43.6%.
=====
```

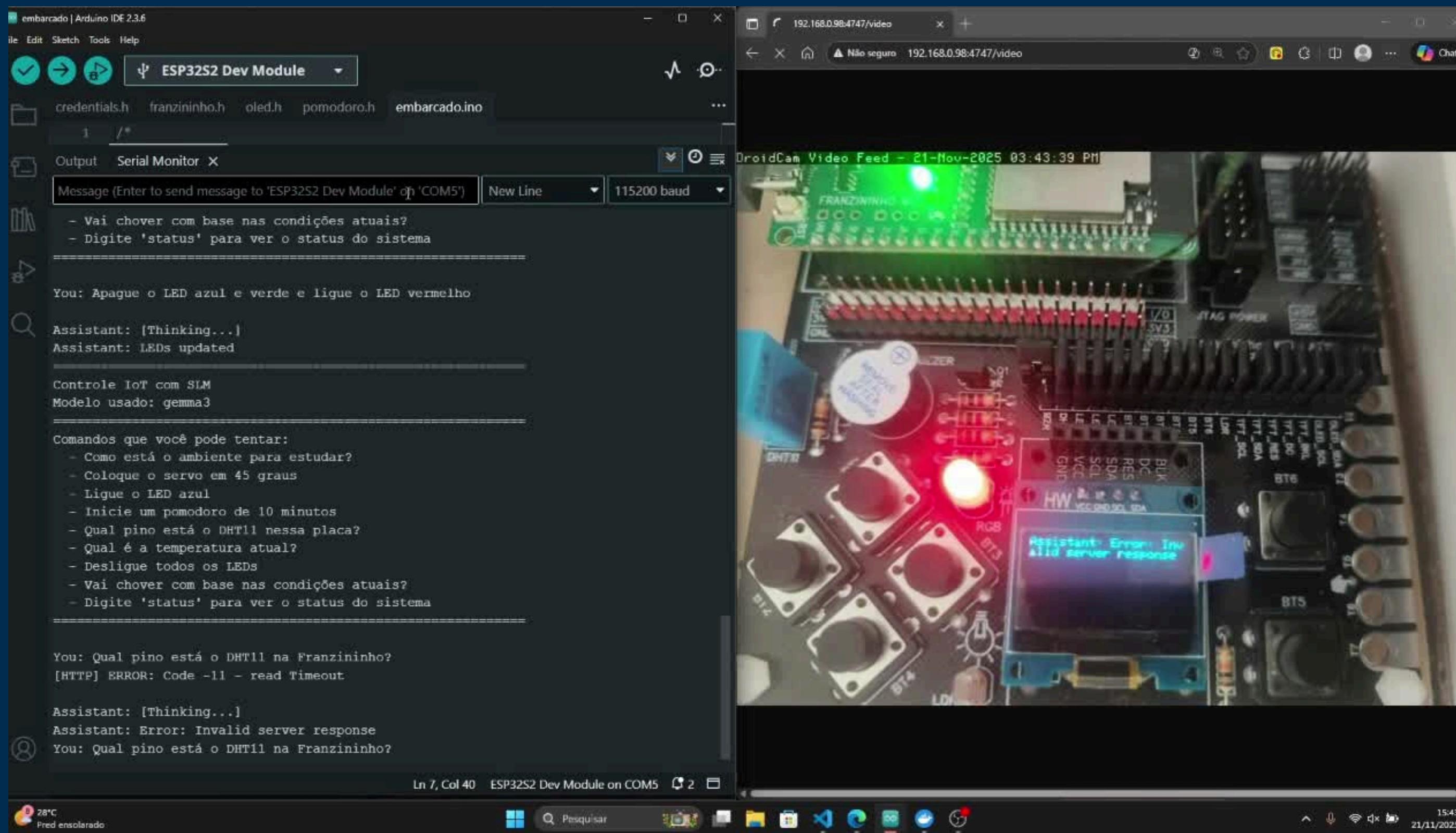


# “Ligue o Led Vermelho”

# “Inicie um pomodoro de 3 minutos”



# “Qual pino está o DHT11 na Franzininho?”



# Reféncias

- Edge AI Engineering Hands-on with the Raspberry Pi
- Franzininho WiFi LAB01 | Franzininho
- Ollama's documentation - Ollama
- Edge AI/SLM IoT Control Local at main · guilhermefernandes/Edge\_AI
- 106 - IA + ESP32: Controle Inteligente com SLM e RAG

# Obrigado



**Guilherme Fernandes de Oliveira**  
**Engenheiro Eletrônico**



**guilhermeferoliv@gmail.com**



**<https://www.linkedin.com/in/iguilherme>**

