

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

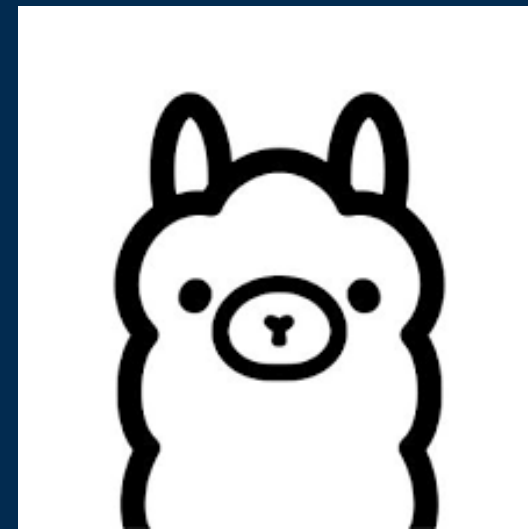
- Rotina intensa na bancada: estudos, soldagem, projetos e ambientes pouco otimizados
- Franzininho WiFi Lab01 com servidor Python e Ollama em Docker
- Uso de SLM + RAG para interpretar comandos e consultar documentação técnica em tempo real

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

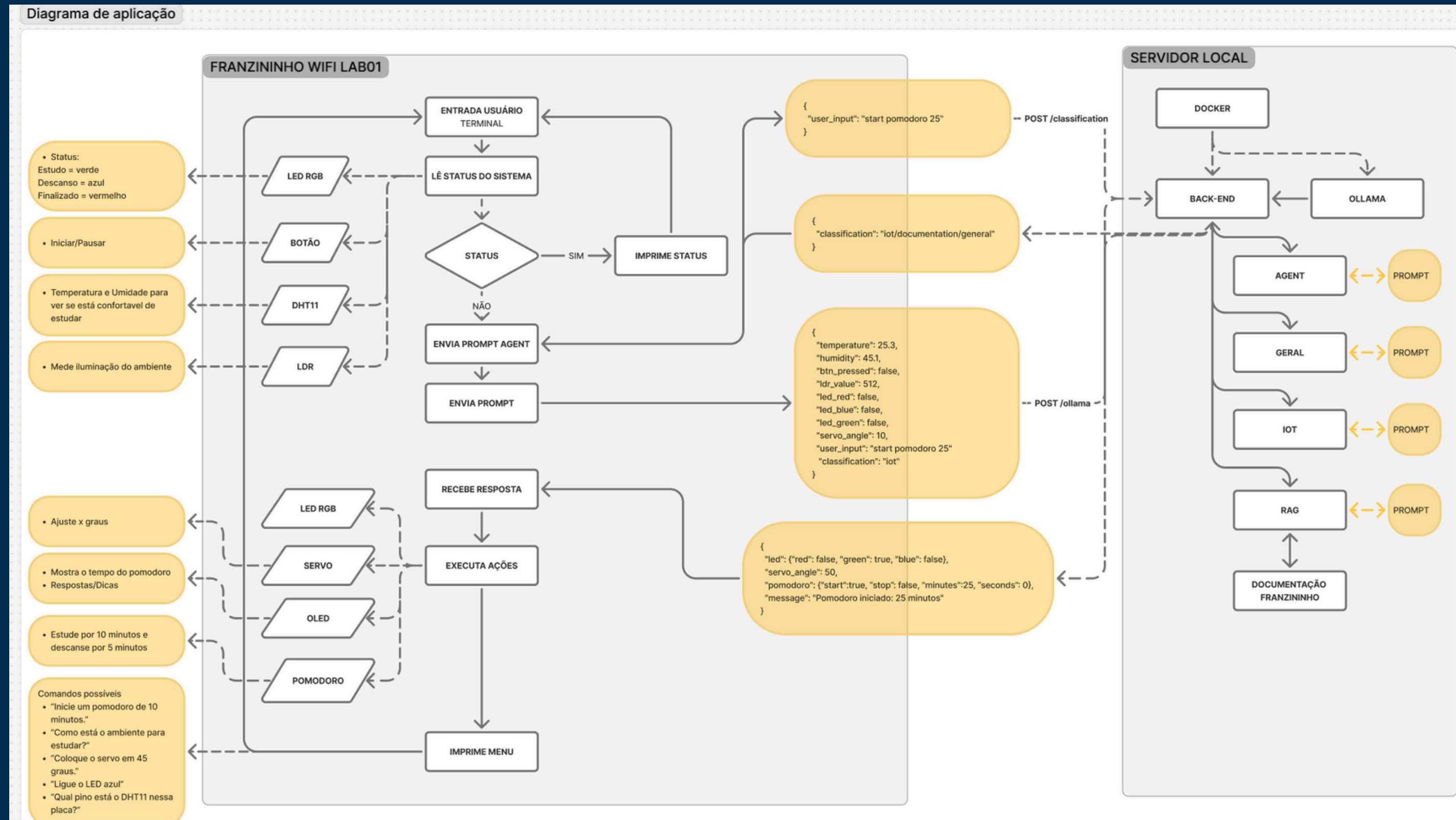


Gemma 3

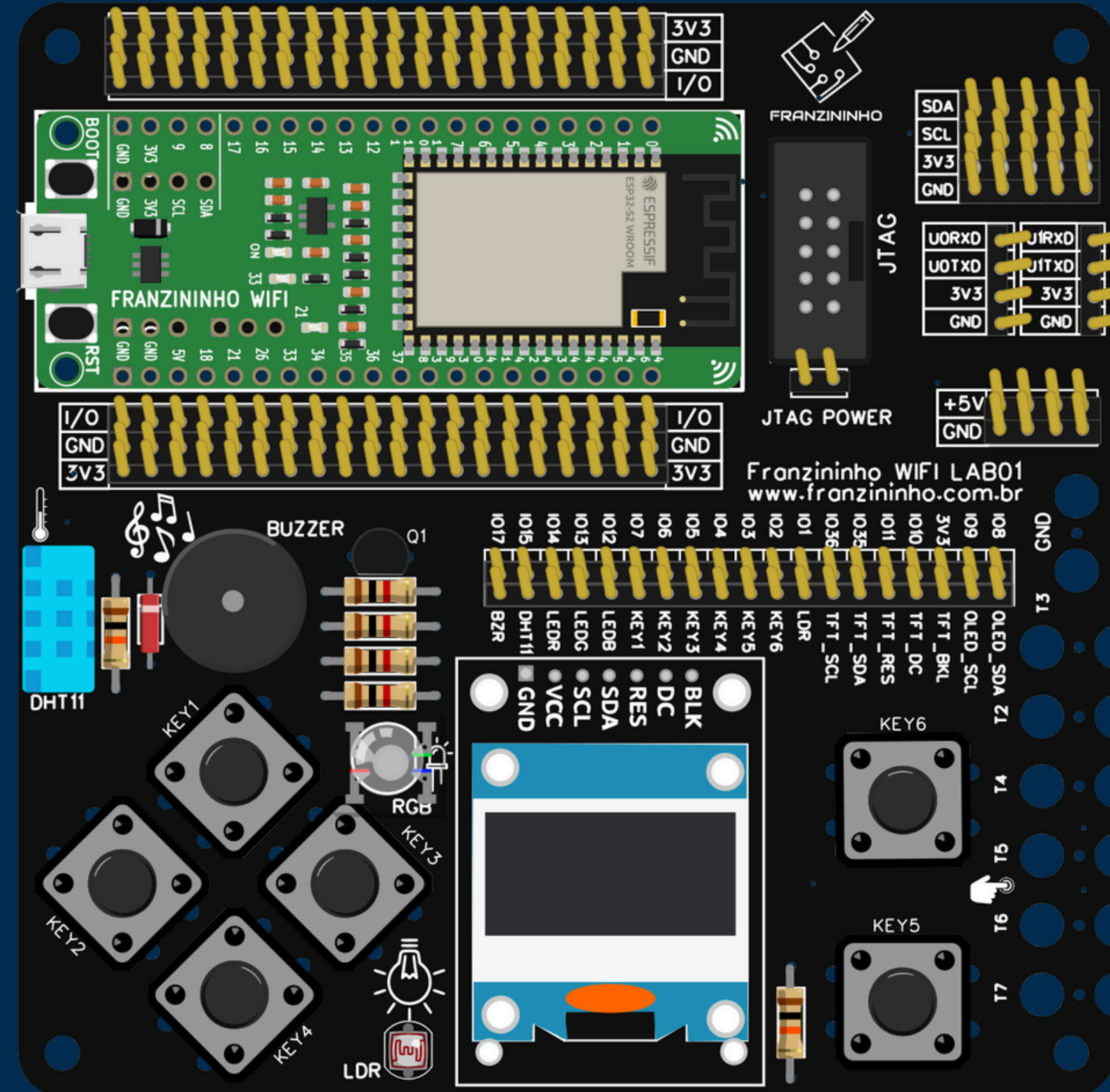


Flask
web development,
one drop at a time

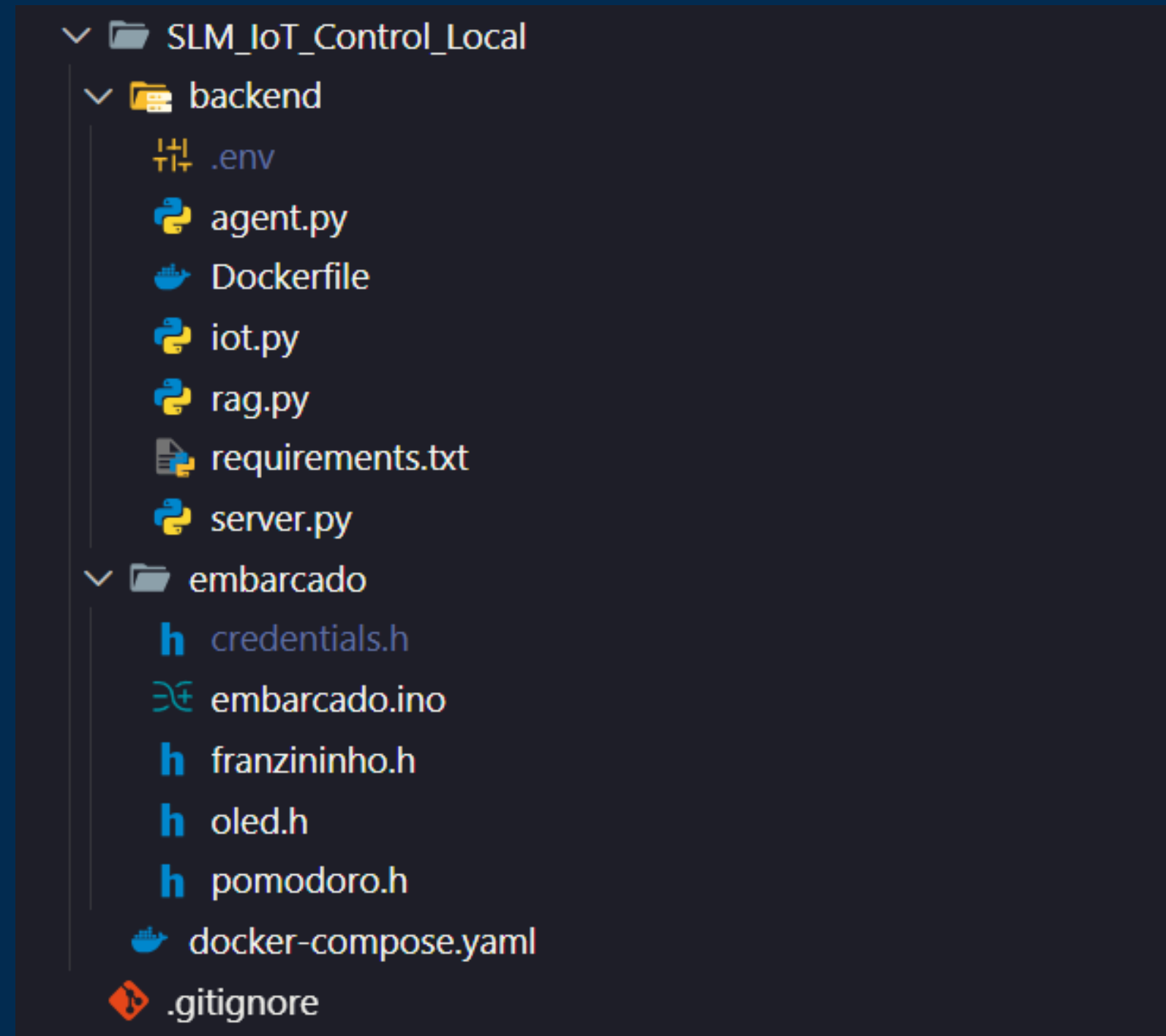
Diagrama da aplicação



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

```
useRAG = RAG(  
    persist_dir="chroma_db",  
    model=MODEL,  
    urls = [  
        "https://raw.githubusercontent.com/Franzininho/docs-franzininho-site/main/docs/FranzininhoWiFiLAB01/franzininho-wifi-lab01.md",  
        "https://docs.franzininho.com.br/docs/franzininho-wifi/franzininho-wifi/"  
    ],  
    pdfs= [],  
    text=[]  
)
```

```
41 class RAG:  
42     def __init__(  
61         self.chunk_size = chunk_size  
62         self.chunk_overlap = chunk_overlap  
63         self.vectorstore = None  
64         self.retriever = None  
65  
66         # self.llm = ChatOllama(model=self.model, temperature=0)  
67  
68         self.preload_models()  
69         self.create_vectorstore(self.urls, self.pdfs, self.text, recreate=False)  
70         self.load_vectorstore()  
71  
72         # -----  
73         # Custom embedding class that uses Ollama directly and implements caching  
74         # -----  
75 > class OptimizedOllamaEmbeddings: ...  
112  
113         # -----  
114         # Model warm-up  
115         # -----  
116 > def preload_models(self): ...  
141  
142         # -----  
143         # Vectorstore creation  
144         # -----  
145 > def create_vectorstore(self, urls=None, pdfs=None, text=None, recreate=False): ...  
219  
220         # -----  
221         # Load existing vectorstore  
222         # -----  
223 > def load_vectorstore(self): ...  
244  
245         # -----  
246         # Query RAG  
247         # -----  
248 > def query(self, question): ...
```

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,

    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): ...

    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: {ldrValue}
- servo_angle: {servoAngle}

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 M SLM_IoT_Control_Local\backend\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.yml({}) services

docker-compose.yml - The Compose specification establishes a standard

```
1 services:
2   ollama:
3     image: ollama/ollama:latest
4     container_name: ollama
5     ports:
6       - "11434:11434"
7     volumes:
8       - ollama_data:/root/.ollama
9     entrypoint: >
10      bash -c "
11      ollama serve &
12      sleep 3 &&
13      if ! ollama list | grep -q 'gemma3'; then
14        echo 'Baixando modelo gemma3...';
15        ollama pull gemma3;
16        ollama pull nomic-embed-text;
17      else
18        echo 'Modelo gemma3 já está instalado.';
19      fi &&
20      wait
21    "
22
23   backend:
24     build:
25       context: ./backend
26       dockerfile: Dockerfile
27     container_name: backend_python
28     ports:
29       - "5000:5000"
30     env_file:
31       - ./backend/.env
32     depends_on:
33       - ollama
34   volumes:
35     ollama_data:
```

“Como está o ambiente para estudar?”

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

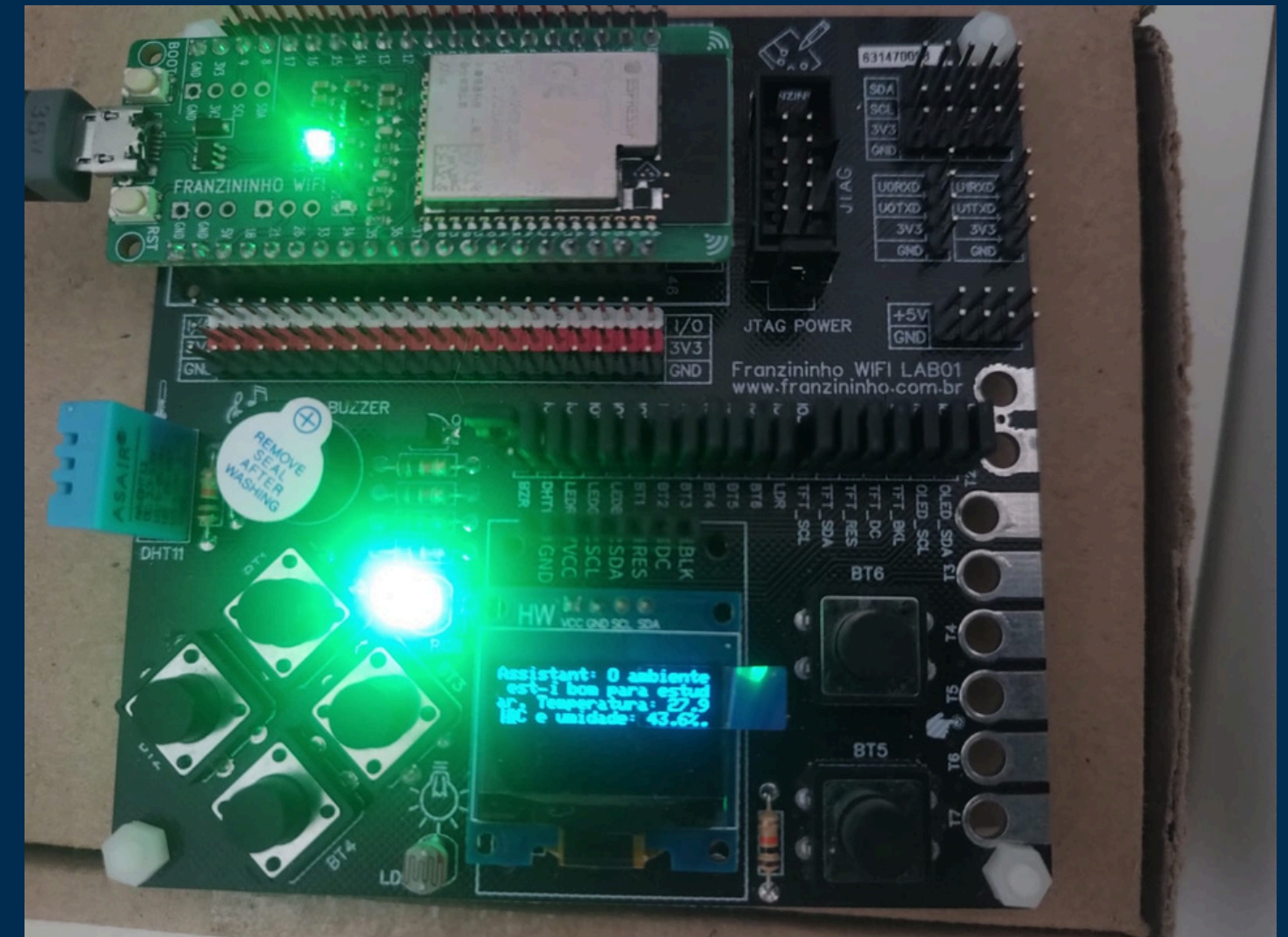
Comandos que você pode tentar:

- 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

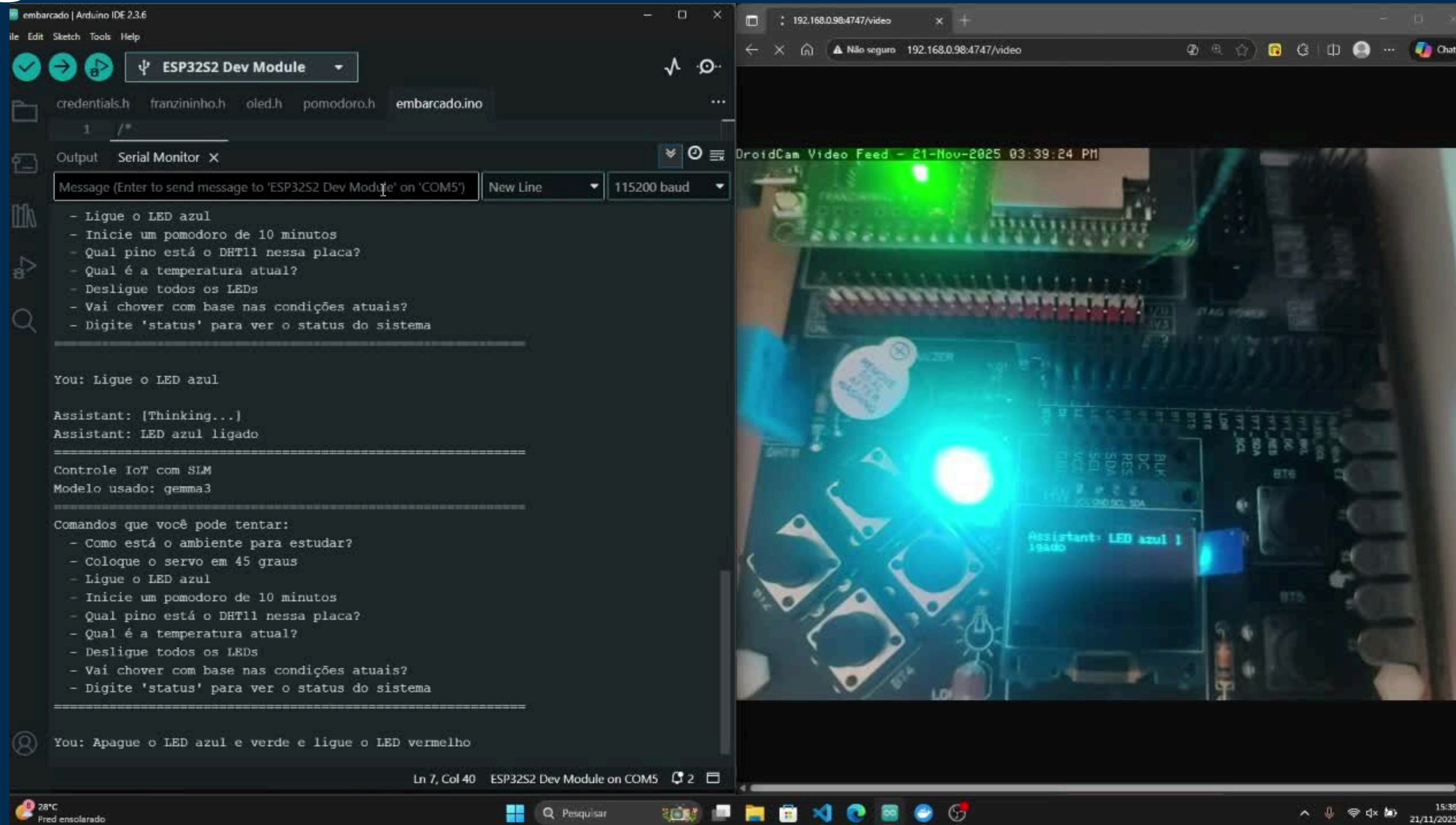
```
=====
You: Como está o ambiente para estudar?
```

```
Assistant: [Thinking...]
```

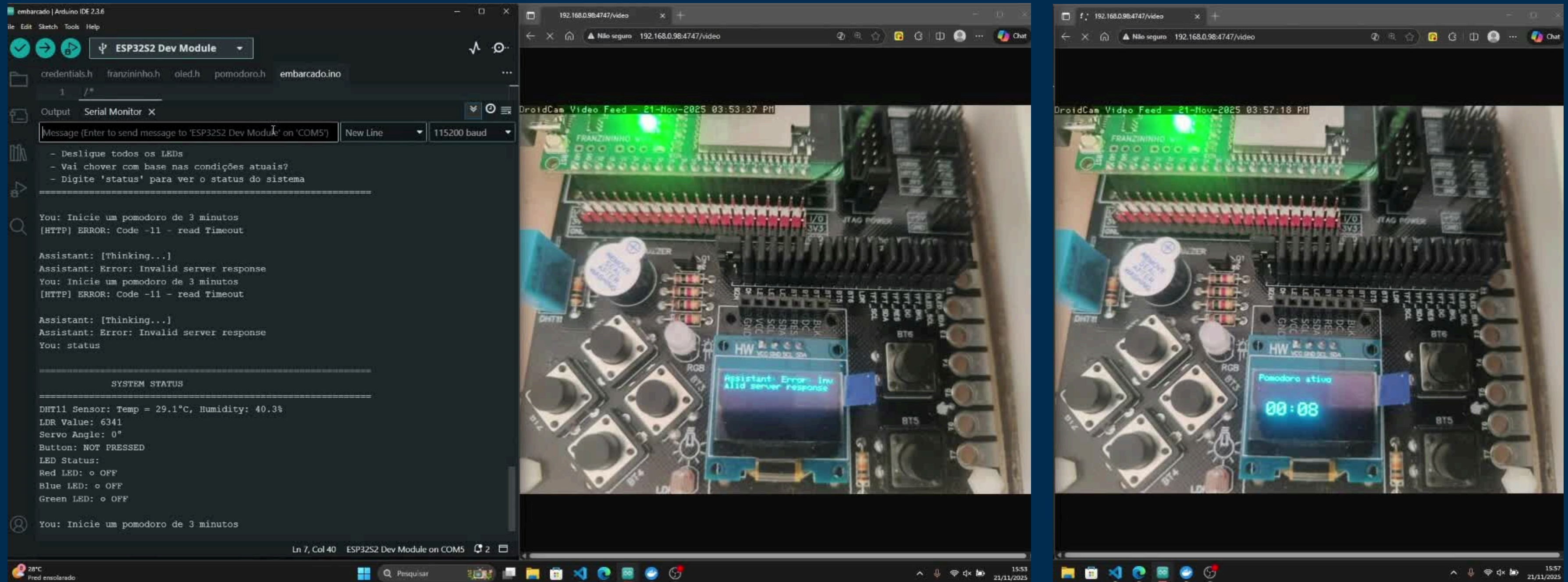
```
Assistant: O ambiente está bom para estudar. Temperatura: 27.9°C e umidade: 43.6%.
=====
```



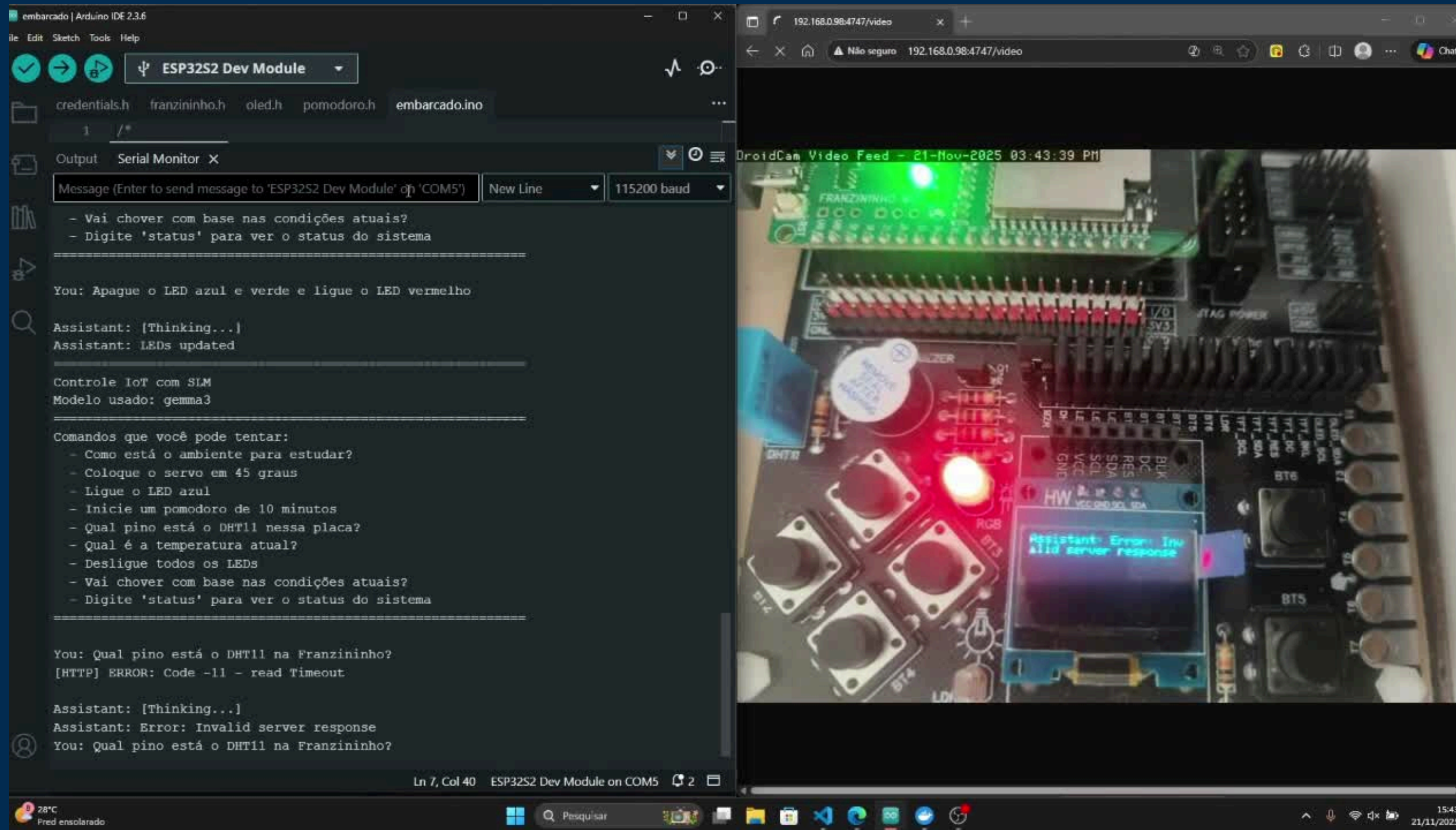
“Ligue o Led Vermelho”



“Inicie um pomodoro de 3 minutos”



“Qual pino está o DHT11 na Franzininho?”



The image shows a dual-screen setup. The left screen displays the Arduino IDE 2.3.6 interface. The top menu bar includes File, Edit, Sketch, Tools, and Help. The toolbar shows icons for checking, running, and uploading code, along with a dropdown menu set to 'ESP32S2 Dev Module'. The file explorer on the left lists 'credentials.h', 'franzininho.h', 'oled.h', 'pomodoro.h', and 'embarcado.ino'. The main editor area shows a code file with a cursor at line 1. The Serial Monitor window is open, displaying a chat interface. The chat history includes a system message about weather and status, a user command to toggle LEDs, an assistant response '[Thinking...]' and 'LEDs updated', a list of commands, and a user query about the DHT11 pin location. The current chat input shows the user asking 'Qual pino está o DHT11 na Franzininho?'. The status bar at the bottom of the IDE indicates 'Ln 7, Col 40' and 'ESP32S2 Dev Module on COM5'.

The right screen shows a web browser window with the address '192.168.0.98:4747/video'. The browser displays a video feed titled 'DroidCam Video Feed - 21-Nov-2025 03:43:39 PM'. The video shows a close-up of a circuit board. A green LED is lit at the top. A red LED is lit in the center. A small LCD screen at the bottom displays the text 'Assistant: Error: Invalid server response' in red. The board has various components, including a DHT11 sensor module, several push buttons, and a USB cable connected to the right side.

Refêrencias

- [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 · guilhermefernandesk/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>

