Specializarea Sisteme Informatice de Monitorizare a Mediului

Grupa 6503

**CarSense - Smart Vehicle Tracker**

Noghi-Cretu Silviu-Antonin

Anul 2025

**Smart Vehicle Tracker**

Silviu-Antonin NOGHI-CREȚU

**Prezentarea temei (Ce?)**

CarSense este o aplicatie desktop inteligenta care permite monitorizarea si gestionarea senzorilor unui vehicul. Scopul ei este de a oferi utilizatorului o imagine clara si centralizata asupra valorilor esentiale ale masinii, cum ar fi temperatura motorului, tensiunea bateriei sau nivelul combustibilului. In cazul in care utilizatorul doreste sa vada unde sau ssub ce conditii s-au activat anumiti senzori, aplicatia returneaza o harta cu locatia precisa a activarii acestora.

Aplicatia este formata din doua parti: un server Flask (API RESTful) care gestioneaza datele si le expune catre client, si o interfata desktop realizata cu customtkinter, care permite interactiunea prietenoasa cu utilizatorul. Datele sunt stocate intr-o baza de date PostgreSQL, si se pot accesa de pe orice calculator care ruleaza clientul.

**Justificarea temei (De ce?)**

Tema a fost aleasa pentru ca imbina elemente moderne de dezvoltare software si are aplicabilitate practica in domeniul auto, digitalizarea vehiculelor devine din ce in ce mai importanta, o astfel de solutie poate fi folosita atat de pasionati de masini, cat si de mecanici sau firme de monitorizare flote.

Am vrut sa construiesc un proiect complet, care sa includa:

* un backend RESTful real, gazduit in cloud
* o interfata grafica prietenoasa pentru utilizator
* integrare cu servicii externe pentru informatii meteo si localizare
* salvarea datelor intr-o baza de date accesibila din orice locatie

In plus, proiectul este usor de extins cu noi tipuri de senzori, functii de analiza sau notificari automate.

**Descrierea functionalitatii proiectului (Cum?)**

Proiectul are doua componente majore si este complet functional in cloud:

1. Backend (server.py - Flask API)

* ofera rute REST pentru adaugare, stergere si listare senzori (/senzori)
* stocheaza valorile intr-o baza PostgreSQL Railway
* valideaza automat valorile in functie de tipul senzorului
* foloseste API-uri externe pentru:
  + identificarea locatiei prin OpenStreetMap (Nominatim)
  + preluarea vremii istorice de la Open-Meteo

2. Frontend (interface.py - GUI customtkinter)

* interfata moderna si intuitiva
* afisare date senzori in timp real
* filtre pentru valori anormale sau pentru anumite tipuri de senzori
* harti generate cu Folium care marcheaza locatia senzorilor
* poate fi compilata in .exe si rulata pe orice Windows fara instalare

3. Hosting si deploy

* tot backendul este gazduit pe Railway la:  
  https://restfull-api-full-project-production.up.railway.app
* baza de date PostgreSQL e tot in Railway, deci totul ruleaza in cloud
* clientul comunica cu serverul prin API public

Bibliografie:

1. *Flask Documentation* – Framework-ul web folosit pentru construirea API-ului: <https://flask.palletsprojects.com/>
2. *Railway App* – Platforma de hosting pentru server si baza de date: <https://www.railway.app/docs>
3. *OpenStreetMap (Nominatim)* – Serviciul de geocodare folosit pentru obtinerea coordonatelor pe baza locatiei: <https://nominatim.openstreetmap.org/>
4. *Open-Meteo API* – API public folosit pentru afisarea vremii istorice: <https://open-meteo.com/>
5. *CustomTkinter* – Biblioteca Python pentru interfete grafice moderne: <https://customtkinter.tomschimansky.com/>
6. *PostgreSQL* – SGBD utilizat pentru stocarea datelor senzorilor: <https://www.postgresql.org/>

**Anexe:**

**Interface.py**

import customtkinter as ctk

from tkinter import messagebox

import requests, random, datetime, webbrowser, os

import folium, matplotlib.pyplot as plt

ctk.set\_appearance\_mode("dark")  # "light", "dark", "system"

ctk.set\_default\_color\_theme("blue")

root = ctk.CTk()

root.title("CarSense - Smart Vehicle Tracker")

root.geometry("1400x700")

CAR\_SENSOR\_TYPES = {

    "engine\_temp":       ("°C", 70, 110),

    "coolant\_temp":      ("°C", 60, 100),

    "oil\_pressure":      ("bar", 1.0, 5.0),

    "battery\_voltage":   ("V", 12.0, 14.8),

    "fuel\_level":        ("%", 0, 100),

    "intake\_air\_temp":   ("°C", -10, 60),

}

COUNTRIES\_COORDINATES = {

    "România": [(43.6, 27.0), (48.2, 21.2)],

    "Germania": [(47.3, 5.9), (55.1, 15.0)],

    "Franța":   [(42.3, -5.1), (51.1, 8.2)],

    "Italia":   [(36.6, 6.6), (47.1, 18.5)]

}

API\_URL = "https://restfull-api-full-project-production.up.railway.app/senzori"

WEATHER\_URL = "https://restfull-api-full-project-production.up.railway.app/senzori/{}/weather"

senzori\_data = []

tabview = ctk.CTkTabview(root, width=1000, height=650)

tabview.pack(padx=0, pady=20, expand=True, fill="both")

main\_tab = tabview.add("Senzori")

# map\_tab = tabview.add("Hartă")  # opțional

# -------------------- Left Controls --------------------

left\_frame = ctk.CTkFrame(main\_tab, width=700)

left\_frame.pack(side="left", fill="y", padx=10, pady=10)

sensors\_scroll = ctk.CTkScrollableFrame(left\_frame, width=700, height=400)

sensors\_scroll.pack(pady=10, fill="both", expand=True)

sensor\_buttons = []

selected\_sensor\_id = None

filter\_tip = ctk.CTkComboBox(left\_frame, values=[""] + list(CAR\_SENSOR\_TYPES.keys()), width=180)

filter\_tip.pack(pady=5)

ctk.CTkButton(left\_frame, text="Filtrează", command=lambda: refresh\_list()).pack()

# -------------------- Right Controls --------------------

right\_frame = ctk.CTkFrame(main\_tab)

right\_frame.pack(side="right", fill="both", expand=True, padx=10, pady=10)

entry\_country = ctk.CTkComboBox(right\_frame, values=list(COUNTRIES\_COORDINATES.keys()))

entry\_country.pack(pady=5)

entry\_country.set("România")

entry\_tip = ctk.CTkComboBox(right\_frame, values=list(CAR\_SENSOR\_TYPES.keys()))

entry\_tip.pack(pady=5)

entry\_tip.set("engine\_temp")

entry\_locatie = ctk.CTkEntry(right\_frame, placeholder\_text="Locație")

entry\_locatie.pack(pady=5)

entry\_time = ctk.CTkEntry(right\_frame)

entry\_time.pack(pady=5)

entry\_marca = ctk.CTkComboBox(right\_frame, values=[], width=180)

entry\_marca.pack(pady=5)

entry\_marca.configure(command=lambda val: load\_models\_for\_make(val))

def delete\_selected\_sensor():

    if selected\_sensor\_id is None:

        messagebox.showwarning("Ștergere", "Nu ai selectat niciun senzor.")

        return

    try:

        requests.delete(f"{API\_URL}/{selected\_sensor\_id}").raise\_for\_status()

        refresh\_list()

    except Exception as e:

        messagebox.showerror("Eroare", f"Eroare la ștergere: {e}")

def update\_time():

    entry\_time.delete(0, 'end')

    entry\_time.insert(0, datetime.datetime.now().isoformat(timespec="seconds"))

    root.after(1000, update\_time)

update\_time()

btn\_frame = ctk.CTkFrame(right\_frame)

btn\_frame.pack(pady=10)

ctk.CTkButton(btn\_frame, text="Adaugă", width=120, command=lambda: add\_sensor()).pack(pady=2)

ctk.CTkButton(btn\_frame, text="Modifică", width=120, command=lambda: update\_sensor()).pack(pady=2)

ctk.CTkButton(btn\_frame, text="Șterge", width=120, command=lambda: delete\_selected\_sensor()).pack(pady=2)

ctk.CTkButton(btn\_frame, text="Istoric", width=120, command=lambda: show\_history\_plot()).pack(pady=2)

ctk.CTkButton(btn\_frame, text="Harta toți", width=120, command=lambda: generate\_map(False)).pack(pady=2)

ctk.CTkButton(btn\_frame, text="Harta probleme", width=120, command=lambda: generate\_map(True)).pack(pady=2)

ctk.CTkButton(btn\_frame, text="Senzor pe hartă", width=120, command=lambda: show\_selected\_sensor\_on\_map()).pack(pady=2)

def refresh\_list():

    global sensor\_buttons, selected\_sensor\_id

    try:

        r = requests.get(API\_URL)

        r.raise\_for\_status()

        data = r.json()

        senzori\_data.clear()

        selected\_sensor\_id = None

        # Curăță frame-ul scrollabil

        for widget in sensors\_scroll.winfo\_children():

            widget.destroy()

        sensor\_buttons.clear()

        selected\_tip = filter\_tip.get()

        for s in data:

            if selected\_tip and s["tip"] != selected\_tip:

                continue

            senzori\_data.append(s)

            vehicul = s.get("vehicul", "")

            if "nicio variantă" in vehicul.lower():

                vehicul = "Necunoscut"

            text = (f"ID: {s['id']} | {s['tip']} = {s['valoare']} {s.get('unitate','')} "

                    f"| Vehicul: {vehicul} | Loc: {s['locatie']}")

            btn = ctk.CTkButton(sensors\_scroll, text=text, width=360, anchor="w", fg\_color="#2a2a2a",

                                command=lambda sid=s['id']: select\_sensor\_by\_id(sid))

            btn.pack(pady=2)

            sensor\_buttons.append(btn)

    except Exception as e:

        messagebox.showerror("Eroare", f"Eroare la refresh: {e}")

def select\_sensor\_by\_id(sid):

    global selected\_sensor\_id

    selected\_sensor\_id = sid

    s = next((sensor for sensor in senzori\_data if sensor["id"] == sid), None)

    if not s:

        return

    entry\_tip.set(s["tip"])

    entry\_locatie.delete(0, 'end')

    entry\_locatie.insert(0, s["locatie"])

    entry\_time.delete(0, 'end')

    entry\_time.insert(0, s["time"])

    vehicul = s.get("vehicul", "")

    parts = vehicul.split(" ", 1)

    if len(parts) == 2:

        marca, model = parts

        entry\_marca.set(marca)

        load\_models\_for\_make(marca)

    else:

        entry\_marca.set("Alege marcă")

def get\_random\_location(country):

    bounds = COUNTRIES\_COORDINATES.get(country)

    if not bounds: return "Unknown"

    lat\_min, lon\_min = bounds[0]; lat\_max, lon\_max = bounds[1]

    lat = round(random.uniform(lat\_min, lat\_max), 5)

    lon = round(random.uniform(lon\_min, lon\_max), 5)

    try:

        r = requests.get("https://nominatim.openstreetmap.org/reverse",

            params={"lat": lat, "lon": lon, "format": "json"},

            headers={"User-Agent": "CarSense-App"}).json()

        return r.get("address", {}).get("city") or r.get("display\_name", f"{lat}, {lon}")

    except: return f"{lat}, {lon}"

def add\_sensor():

    try:

        tip = entry\_tip.get()

        unit, mn, mx = CAR\_SENSOR\_TYPES[tip]

        loc = get\_random\_location(entry\_country.get())

        vehicul = f"{entry\_marca.get()}"

        if "nicio variantă" in vehicul.lower():

            vehicul = entry\_marca.get()

        data = {

            "tip": tip,

            "valoare": int(random.uniform(mn, mx)) if tip == "fuel\_level" else round(random.uniform(mn, mx), 2),

            "locatie": loc,

            "vehicul": vehicul,

            "time": entry\_time.get()

        }

        r = requests.post(API\_URL, json=data)

        r.raise\_for\_status()

        response\_data = r.json()

        refresh\_list()

    except Exception as e:

        messagebox.showerror("Eroare", str(e))

def load\_makes():

    try:

        r = requests.get("https://carapi.app/api/makes")

        r.raise\_for\_status()

        data = r.json().get("data", [])

        makes = sorted({

            item["name"] for item in data

            if item.get("name") and "subscription required" not in item["name"].lower()

        })

        if makes:

            entry\_marca.configure(values=list(makes))

            entry\_marca.set(makes[0])

            load\_models\_for\_make(makes[0])

        else:

            entry\_marca.configure(values=["Nicio marcă disponibilă"])

            entry\_marca.set("Nicio marcă disponibilă")

    except Exception as e:

        messagebox.showerror("Eroare", f"Nu s-au putut încărca mărcile: {e}")

# La începutul fișierului

model\_cache = {}

def load\_models\_for\_make(marca):

    try:

        if marca in model\_cache:

            models = model\_cache[marca]

        else:

            r = requests.get("https://carapi.app/api/models", params={"make": marca})

            r.raise\_for\_status()

            raw\_models = r.json().get("data", [])

            models = sorted({

                item["name"] for item in raw\_models

                if item.get("name") and "subscription required" not in item["name"].lower()

            })

            model\_cache[marca] = models

    except Exception as e:

        messagebox.showerror("Eroare", f"Nu s-au putut încărca modelele: {e}")

def update\_sensor():

    if selected\_sensor\_id is None:

        messagebox.showwarning("Modificare", "Selectează un senzor din listă mai întâi.")

        return

    try:

        sensor = next((s for s in senzori\_data if s["id"] == selected\_sensor\_id), None)

        if not sensor:

            messagebox.showerror("Eroare", "Senzorul nu a fost găsit în listă.")

            return

        vehicul = f"{entry\_marca.get()}"

        if "nicio variantă" in vehicul.lower():

            vehicul = entry\_marca.get()

        updated\_data = {

            "id": sensor["id"],

            "tip": entry\_tip.get(),

            "valoare": sensor["valoare"],  # păstrăm valoarea originală

            "locatie": entry\_locatie.get(),

            "vehicul": vehicul,

            "time": entry\_time.get()

        }

        r = requests.post(API\_URL, json=updated\_data)

        r.raise\_for\_status()

        refresh\_list()

        messagebox.showinfo("Succes", f"Senzorul #{sensor['id']} a fost actualizat.")

    except Exception as e:

        messagebox.showerror("Eroare", f"Eroare la modificare: {e}")

def show\_history\_plot():

    if not senzori\_data: return

    s = senzori\_data[-1]  # Ultimul senzor din listă

    tip = s['tip']; valoare = float(s['valoare'])

    unit = CAR\_SENSOR\_TYPES.get(tip, ("",))[0]

    valori = [round(valoare + random.uniform(-5, 5), 2) for \_ in range(10)]

    timp = [f"T-{i}" for i in range(10, 0, -1)]

    plt.figure(figsize=(8, 4))

    plt.plot(timp, valori, marker='o')

    plt.title(f"Istoric pentru '{tip}'")

    plt.xlabel("Timp"); plt.ylabel(f"Valoare ({unit})")

    plt.grid(True); plt.tight\_layout(); plt.show()

def generate\_map(only\_problems=False):

    m = folium.Map(location=[45.9, 24.9], zoom\_start=6)

    coduri = {

        0: "Cer senin",

        1: "Parțial noros",

        2: "Noros",

        3: "Înnorat",

        45: "Ceață",

        48: "Ceață cu depuneri de gheață",

        51: "Burniță slabă",

        53: "Burniță moderată",

        55: "Burniță intensă",

        56: "Burniță înghețată slabă",

        57: "Burniță înghețată intensă",

        61: "Ploaie slabă",

        63: "Ploaie moderată",

        65: "Ploaie intensă",

        66: "Ploaie înghețată slabă",

        67: "Ploaie înghețată intensă",

        71: "Ninsoare slabă",

        73: "Ninsoare moderată",

        75: "Ninsoare abundentă",

        77: "Fulgi de zăpadă",

        80: "Averse slabe",

        81: "Averse moderate",

        82: "Averse puternice",

        85: "Averse de ninsoare slabă",

        86: "Averse de ninsoare intensă",

        95: "Furtună",

        96: "Furtună cu grindină slabă",

        99: "Furtună cu grindină severă"

    }

    limits = {

        "engine\_temp":     lambda v: v > 105,

        "coolant\_temp":    lambda v: v > 95,

        "oil\_pressure":    lambda v: v < 1.5,

        "battery\_voltage": lambda v: v < 12.2,

        "fuel\_level":      lambda v: v < 10,

        "intake\_air\_temp": lambda v: v > 50,

    }

    added = 0

    for s in senzori\_data:

        if only\_problems and not limits.get(s["tip"], lambda \_: False)(s["valoare"]):

            continue

        try:

            geo = requests.get("https://nominatim.openstreetmap.org/search",

                params={"q": s["locatie"], "format": "json", "limit": 1},

                headers={"User-Agent": "CarSense-App"}).json()

            if not geo:

                continue

            lat, lon = float(geo[0]["lat"]), float(geo[0]["lon"])

            w = requests.get(WEATHER\_URL.format(s["id"])).json()

            cod = w.get("weathercode", "-")

            vehicul = s.get('vehicul', '')

            if "nicio variantă" in vehicul.lower():

                vehicul = "Necunoscut"

            popup = folium.Popup(f"""

            <div style='width: 260px; font-size: 13px; line-height: 1.5; white-space: normal;'>

            <b>Tip senzor:</b> {s['tip']}<br>

            <b>Valoare:</b> {s['valoare']} {s.get('unitate','')}<br>

            <b>Vehicul:</b> {vehicul}<br>

            <b>Locație:</b> {s['locatie']}<br>

            <b>Data:</b> {s['time']}<br>

            <b>Vreme:</b> {coduri.get(cod, 'necunoscut')} (cod {cod})<br>

            <b>Temp. exterioară:</b> {w.get("temperature\_2m", "N/A")} °C

            </div>

            """, max\_width=300)

            color = "red" if limits.get(s["tip"], lambda \_: False)(s["valoare"]) else "green"

            folium.Marker([lat, lon], popup=popup, icon=folium.Icon(color=color)).add\_to(m)

            added += 1

        except:

            continue

    if added == 0:

        messagebox.showwarning("Harta", "Nicio locație validă.")

        return

    m.save("map.html")

    webbrowser.open(f"file://{os.path.abspath('map.html')}")

def show\_selected\_sensor\_on\_map():

    if selected\_sensor\_id is None:

        messagebox.showwarning("Harta", "Nu ai selectat niciun senzor.")

        return

    sensor = next((s for s in senzori\_data if s["id"] == selected\_sensor\_id), None)

    if not sensor:

        return

    loc = sensor.get("locatie", "")

    try:

        geo = requests.get("https://nominatim.openstreetmap.org/search",

            params={"q": loc, "format": "json", "limit": 1},

            headers={"User-Agent": "CarSense-App"}).json()

        if not geo:

            messagebox.showerror("Eroare", "Locația nu a putut fi geocodificată.")

            return

        lat, lon = float(geo[0]["lat"]), float(geo[0]["lon"])

        # Obține vremea

        w = requests.get(WEATHER\_URL.format(sensor["id"])).json()

        cod = w.get("weathercode", "-")

        coduri = {

            0: "Cer senin", 1: "Parțial noros", 2: "Noros", 3: "Înnorat",

            45: "Ceață", 48: "Ceață cu depuneri", 51: "Burniță slabă", 53: "Burniță moderată", 55: "Burniță puternică",

            56: "Burniță înghețată slabă", 57: "Burniță înghețată intensă", 61: "Ploaie slabă", 63: "Ploaie moderată",

            65: "Ploaie puternică", 66: "Ploaie înghețată slabă", 67: "Ploaie înghețată intensă",

            71: "Ninsoare slabă", 73: "Ninsoare moderată", 75: "Ninsoare puternică", 77: "Fulgi de zăpadă",

            80: "Averse slabe", 81: "Averse moderate", 82: "Averse puternice",

            85: "Averse de ninsoare slabă", 86: "Averse de ninsoare intensă",

            95: "Furtună", 96: "Furtună cu grindină", 99: "Furtună severă"

        }

        vehicul = sensor.get('vehicul', 'Necunoscut')

        if "nicio variantă" in vehicul.lower():

            vehicul = "Necunoscut"

        popup = folium.Popup(f"""

            <div style='width: 260px; font-size: 13px; line-height: 1.5; white-space: normal;'>

            <b>Tip senzor:</b> {sensor['tip']}<br>

            <b>Valoare:</b> {sensor['valoare']} {sensor.get('unitate','')}<br>

            <b>Vehicul:</b> {vehicul}<br>

            <b>Locație:</b> {sensor['locatie']}<br>

            <b>Data:</b> {sensor['time']}<br>

            <b>Vreme:</b> {coduri.get(cod, 'necunoscut')} (cod {cod})<br>

            <b>Temp. exterioară:</b> {w.get("temperature\_2m", "N/A")} °C

            </div>

        """, max\_width=300)

        m = folium.Map(location=[lat, lon], zoom\_start=12)

        folium.Marker([lat, lon], popup=popup, icon=folium.Icon(color="blue")).add\_to(m)

        m.save("selected.html")

        webbrowser.open(f"file://{os.path.abspath('selected.html')}")

    except Exception as e:

        messagebox.showerror("Eroare", f"Eroare la afișarea pe hartă: {e}")

refresh\_list()

load\_makes()

root.mainloop()

#--------------------------------------------------------------------------------------#

**Server.py**

from flask import Flask, request, jsonify

import psycopg2

import os

import requests

from datetime import datetime

app = Flask(\_\_name\_\_)

# Conexiune PostgreSQL (Railway)

DATABASE\_URL = os.getenv("DATABASE\_URL",

"postgresql://postgres:pDSmlXNxsFYsBnNkbryBUxaorrMjdVbs@centerbeam.proxy.rlwy.net:49284/railway"

)

conn = psycopg2.connect(DATABASE\_URL, sslmode='require')

cursor = conn.cursor()

# Tipuri de senzori auto

CAR\_SENSOR\_TYPES = {

"engine\_temp": {"unit": "°C", "min": 70, "max": 110},

"coolant\_temp": {"unit": "°C", "min": 60, "max": 100},

"oil\_pressure": {"unit": "bar", "min": 1.0, "max": 5.0},

"battery\_voltage": {"unit": "V", "min": 12.0,"max": 14.8},

"fuel\_level": {"unit": "%", "min": 0, "max": 100},

"intake\_air\_temp": {"unit": "°C", "min": -10, "max": 60}

}

cursor.execute("""

CREATE TABLE IF NOT EXISTS senzori2 (

id SERIAL PRIMARY KEY,

tip TEXT NOT NULL,

valoare REAL NOT NULL,

locatie TEXT NOT NULL,

vehicul TEXT,

time TEXT NOT NULL

);

""")

cursor.execute("""

ALTER TABLE senzori2

ADD COLUMN IF NOT EXISTS vehicul TEXT;

""")

conn.commit()

@app.route("/senzori", methods=["GET"])

def get\_sensors():

cursor.execute("SELECT id, tip, valoare, locatie, vehicul, time FROM senzori2;")

rows = cursor.fetchall()

sensors = []

for r in rows:

tip = r[1]

unit = CAR\_SENSOR\_TYPES.get(tip, {}).get("unit", "")

sensors.append({

"id": r[0],

"tip": tip,

"valoare": r[2],

"unitate": unit,

"locatie": r[3],

"vehicul": r[4],

"time": r[5]

})

return jsonify(sensors), 200

@app.route("/senzori", methods=["POST"])

def add\_sensor():

data = request.json or {}

tip = data.get("tip")

valoare = data.get("valoare")

if tip not in CAR\_SENSOR\_TYPES:

return jsonify({"error": "Tip de senzor invalid"}), 400

conf = CAR\_SENSOR\_TYPES[tip]

if valoare is None or not (conf["min"] <= valoare <= conf["max"]):

return jsonify({"error": f"Valoare în afara limitelor ({conf['min']}–{conf['max']})"}), 400

try:

cursor.execute("""

INSERT INTO senzori2 (tip, valoare, locatie, vehicul, time)

VALUES (%s, %s, %s, %s, %s)

RETURNING id;

""", (

tip, valoare,

data.get("locatie"), data.get("vehicul"), data.get("time")

))

new\_id = cursor.fetchone()[0]

conn.commit()

return jsonify({"message": "Senzor adăugat", "id": new\_id}), 201

except Exception as e:

conn.rollback()

print("Eroare la adăugare senzor:", e) # Asta ajută la debugging

return jsonify({"error": str(e)}), 500

@app.route("/senzori/<int:sensor\_id>", methods=["DELETE"])

def delete\_sensor(sensor\_id):

try:

cursor.execute("DELETE FROM senzori2 WHERE id = %s;", (sensor\_id,))

conn.commit()

return jsonify({"message": "Senzor șters"}), 202

except Exception as e:

conn.rollback()

return jsonify({"error": str(e)}), 500

@app.route("/senzori/tipuri", methods=["GET"])

def get\_types():

return jsonify(CAR\_SENSOR\_TYPES), 200

@app.route("/geocode", methods=["GET"])

def geocode():

lat = request.args.get("lat")

lon = request.args.get("lon")

if not lat or not lon:

return jsonify({"error": "Trebuie specificați parametri lat și lon"}), 400

resp = requests.get(

"https://nominatim.openstreetmap.org/reverse",

params={"format": "jsonv2", "lat": lat, "lon": lon, "accept-language": "ro"},

headers={"User-Agent": "CarSense-App"}

)

data = resp.json()

addr = data.get("address", {})

city = addr.get("city") or addr.get("town") or addr.get("village") or addr.get("county")

return jsonify({"lat": float(lat), "lon": float(lon), "city": city}), 200

@app.route("/senzori/<int:sensor\_id>/weather", methods=["GET"])

def sensor\_weather(sensor\_id):

cursor.execute("SELECT locatie, time FROM senzori2 WHERE id = %s;", (sensor\_id,))

row = cursor.fetchone()

if not row:

return jsonify({"error": "Senzor nu există"}), 404

locatie, timestr = row

geo = requests.get(

"https://nominatim.openstreetmap.org/search",

params={"q": locatie, "format": "jsonv2", "limit": 1},

headers={"User-Agent": "CarSense-App"}

).json()

if not geo:

return jsonify({"error": "Localitate necunoscută"}), 400

lat, lon = geo[0]["lat"], geo[0]["lon"]

dt = datetime.fromisoformat(timestr)

date = dt.date().isoformat()

hour\_str = f"{dt.hour:02d}:00"

params = {

"latitude": lat, "longitude": lon,

"start\_date": date, "end\_date": date,

"hourly": "temperature\_2m,relativehumidity\_2m,precipitation,weathercode,winddirection\_10m,windspeed\_10m",

"timezone": "Europe/Bucharest"

}

wresp = requests.get("https://api.open-meteo.com/v1/forecast", params=params)

if wresp.status\_code != 200:

return jsonify({"error": "Eroare la API-ul meteo"}), 502

hourly = wresp.json().get("hourly", {})

times = hourly.get("time", [])

try:

idx = times.index(f"{date}T{hour\_str}")

except ValueError:

return jsonify({"error": "Ora nu e disponibilă în răspuns"}), 502

return jsonify({

"sensor\_id": sensor\_id,

"location": locatie,

"datetime": timestr,

"latitude": lat,

"longitude": lon,

"temperature\_2m": hourly["temperature\_2m"][idx],

"humidity": hourly["relativehumidity\_2m"][idx],

"precipitation": hourly["precipitation"][idx],

"weathercode": hourly["weathercode"][idx],

"winddirection\_10m":hourly["winddirection\_10m"][idx],

"windspeed\_10m": hourly["windspeed\_10m"][idx]

}), 200

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host="0.0.0.0", port=5000)