```python
from flask import Flask, request, jsonify
from flask_cors import CORS
import joblib
import pandas as pd
import requests
from datetime import datetime
import unicodedata
import mysql.connector

from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives import serialization
import base64

# Inicialização do app Flask
app = Flask(__name__)
CORS(app)

# Carrega o modelo
modelo = joblib.load("modelo_preco.pkl")

# Chave da API Google (ATENÇÃO: mantenha segura em produção!)
GOOGLE_API_KEY = "AIzaSyDLZuBuKwtf5kIBRrC7e_Yf3Qaf8RuWi10"

# Configurações do banco de dados
DB_CONFIG = {
    "host": "doamaisbd.mysql.database.azure.com",
    "user": "doamaisadmin",
    "password": "ChargeBack2nads",
    "database": "db_pick_your_driver",
    "ssl_disabled": False
}

# --- CARREGA CHAVE PÚBLICA PARA RSA ---
with open("public_key.pem", "rb") as f:
    public_key = serialization.load_pem_public_key(f.read())

def criptografar_rsa(mensagem):
    """Criptografa uma string usando a chave pública RSA e retorna base64."""
    criptografado = public_key.encrypt(
        mensagem.encode("utf-8"),
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return base64.b64encode(criptografado).decode("utf-8")

# Utilitários
def limpar_endereco(endereco):
    nfkd = unicodedata.normalize('NFKD', endereco)
    return u"".join([c for c in nfkd if not unicodedata.combining(c)])

def endereco_para_coordenadas(endereco):
    endereco_limpo = limpar_endereco(endereco)
    url = f"https://maps.googleapis.com/maps/api/geocode/json?address={endereco_limpo}&key={GOOGLE_API_KEY}"
    response = requests.get(url)
    data = response.json()

    if data['status'] == 'OK':
        location = data['results'][0]['geometry']['location']
        return location['lat'], location['lng']
    else:
        raise ValueError(f"Erro ao geocodificar com Google: {data['status']}")

def calcular_distancia_google(origem, destino):
    lat1, lng1 = endereco_para_coordenadas(origem)
    lat2, lng2 = endereco_para_coordenadas(destino)

    url = (
        f"https://maps.googleapis.com/maps/api/distancematrix/json?"
        f"origins={lat1},{lng1}&destinations={lat2},{lng2}&key={GOOGLE_API_KEY}"
    )
    response = requests.get(url)
    data = response.json()

    if data['status'] == 'OK':
        elemento = data['rows'][0]['elements'][0]
        if elemento['status'] == 'OK':
            distancia_km = elemento['distance']['value'] / 1000
```

```
            tempo_min = elemento['duration']['value'] / 60
            return lat1, lng1, lat2, lng2, distancia_km, tempo_min
        else:
            raise ValueError(f"Erro na Distance Matrix: {elemento['status']}")
    else:
        raise ValueError(f"Erro na API Distance Matrix: {data['status']}")

def gerar_features(end_origem, end_destino):
    lat1, lng1, lat2, lng2, distancia, tempo = calcular_distancia_google(end_origem, end_destino)

    now = datetime.now()
    schedule_time = now.hour * 3600 + now.minute * 60 + now.second
    dia = now.weekday()

    orig_dest_x = abs(hash(end_origem)) % (10 ** 8)
    orig_dest_y = abs(hash(end_destino)) % (10 ** 8)

    df = pd.DataFrame([{
        "OrigDest_x": orig_dest_x,
        "Lat1": lat1,
        "Lng1": lng1,
        "OrigDest_y": orig_dest_y,
        "Lat2": lat2,
        "Lng2": lng2,
        "Distancia": distancia,
        "Dia": dia,
        "schedule_time": schedule_time
    }])

    return df, distancia, tempo, lat1, lng1, lat2, lng2

def salvar_viagem_banco(nome, email, origem, destino, lat1, lng1, lat2, lng2, distancia, tempo, preco):
    try:
        conexao = mysql.connector.connect(**DB_CONFIG)
        cursor = conexao.cursor()
        query = """
            INSERT INTO viagens (
                nome, email, endereco_partida, endereco_destino,
                latitude_partida, longitude_partida,
                latitude_destino, longitude_destino,
                distancia_kilometros, tempo_estimado, preco_estimado
            ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
        """
        valores = (
            nome, email, origem, destino,
            lat1, lng1, lat2, lng2,
            round(distancia, 2), f"{round(tempo, 1)} min", round(preco, 2)
        )
        cursor.execute(query, valores)
        conexao.commit()
        cursor.close()
        conexao.close()
    except Exception as e:
        print(f"Erro ao salvar no banco: {e}")

@app.route("/api/prever", methods=["POST"])
def prever_preco():
    try:
        dados = request.get_json()
        nome = dados.get("nome", "Cliente Anônimo")
        email = dados.get("email", "sem_email@exemplo.com")
        origem = dados.get("endereco_partida", "Avenida Paulista, São Paulo, SP")
        destino = dados.get("endereco_destino", "Praça da Sé, São Paulo, SP")

        # Criptografa nome e email usando RSA
        nome = criptografar_rsa(nome)
        email = criptografar_rsa(email)

        features, distancia, tempo, lat1, lng1, lat2, lng2 = gerar_features(origem, destino)
        preco = modelo.predict(features)[0]

        salvar_viagem_banco(nome, email, origem, destino, lat1, lng1, lat2, lng2, distancia, tempo, preco)

        return jsonify({
            "preco_ubex": round(preco, 2),
            "preco_confort": round(preco * 1.25, 2),
            "preco_black": round(preco * 1.25 * 1.24, 2),
            "distancia_km": round(distancia, 2),
            "tempo_estimado_min": round(tempo, 1)
        })

    except Exception as e:
        return jsonify({"erro": str(e)}), 500
```

```
165
166 if __name__ == "__main__":
167     app.run(debug=True)
168
```

Analise arquivos com o Gemini