Backend Project

<https://colab.research.google.com/notebooks/intro.ipynb>

mport smtplib

from datetime import datetime

from email.mime.text import MIMEText

import pandas as pd

import random

import re

# Load the dataset

df = pd.read\_csv('zocdoc.csv')

# Dictionary to store appointments (date -> time -> doctor)

appointments = {}

# Dictionary to track user's appointments

user\_appointments = {}

# Dictionary to store doctors by specialty

doctors\_by\_specialty = df.groupby('speciality')['Doctor\'s Name'].apply(list).to\_dict()

# Mapping of symptoms to specialties

symptom\_to\_specialty = {

    "fever": "General Physician",

    "cough": "Pulmonologist",

    "chest pain": "Cardiologist",

    "skin rash": "Dermatologist",

    "eye pain": "Ophthalmologist",

    "toothache": "Dentist",

    "joint pain": "Orthopedic",

    "stomach pain": "Gastroenterologist"

}

def send\_email(to\_email, subject, body):

    sender\_email = "[doctorappointmentchatbot@gmail.com](mailto:doctorappointmentchatbot@gmail.com)"

    password = "tpvb nebt xkak azog"

    msg = MIMEText(body)

    msg["Subject"] = subject

    msg["From"] = sender\_email

    msg["To"] = to\_email

    try:

        with smtplib.SMTP\_SSL("[smtp.gmail.com](http://smtp.gmail.com/)", 465) as server:

            server.login(sender\_email, password)

            server.sendmail(sender\_email, to\_email, msg.as\_string())

        print("Email sent successfully!")

    except Exception as e:

        print("Error sending email:", e)

def validate\_mobile(mobile):

    return re.match(r'^\d{10}$', mobile) is not None

def parse\_date(date\_str):

    for fmt in ("%Y-%m-%d", "%d-%m-%Y", "%m/%d/%Y", "%d/%m/%Y"):

        try:

            return datetime.strptime(date\_str, fmt).strftime("%Y-%m-%d")

        except ValueError:

            continue

    return None

def book\_appointment():

    name = input("Enter your name: ")

    email = input("Enter your email: ")

    mobile = input("Enter your mobile number: ")

    while not validate\_mobile(mobile):

        print("Mobile number must be 10 digits.")

        mobile = input("Enter your mobile number: ")

    age = input("Enter your age: ")

    gender = input("Enter your gender: ")

    symptoms = input("Enter your symptoms: ").lower()

    # Suggest specialty based on symptoms

    suggested\_specialty = symptom\_to\_specialty.get(symptoms, "General Physician")

    print(f"Based on your symptoms, we recommend a {suggested\_specialty}.")

    if suggested\_specialty not in doctors\_by\_specialty:

        print("No doctors available for the suggested specialty.")

        return

    doctors = doctors\_by\_specialty[suggested\_specialty]

    print("\nAvailable doctors:")

    for idx, doctor in enumerate(doctors):

        print(f"{idx + 1}. {doctor}")

    doctor\_choice = int(input("Select a doctor: ")) - 1

    if doctor\_choice < 0 or doctor\_choice >= len(doctors):

        print("Invalid selection!")

        return

    selected\_doctor = doctors[doctor\_choice]

    appointment\_date = input("Enter appointment date (YYYY-MM-DD or DD-MM-YYYY or MM/DD/YYYY or DD/MM/YYYY): ")

    parsed\_date = parse\_date(appointment\_date)

    while not parsed\_date:

        print("Invalid date format. Please use YYYY-MM-DD, DD-MM-YYYY, MM/DD/YYYY, or DD/MM/YYYY.")

        appointment\_date = input("Enter appointment date: ")

        parsed\_date = parse\_date(appointment\_date)

    appointment\_time = input("Enter appointment time (HH:MM AM/PM): ")

    today = datetime.today().strftime("%Y-%m-%d")

    if parsed\_date < today:

        print("Cannot book an appointment for a past date!")

        return

    # Check if time slot is already booked

    while parsed\_date in appointments and appointment\_time in appointments[parsed\_date]:

        print("This time slot is already booked!")

        appointment\_time = input("Please enter another appointment time: ")

    if parsed\_date not in appointments:

        appointments[parsed\_date] = {}

    appointments[parsed\_date][appointment\_time] = selected\_doctor

    if email not in user\_appointments:

        user\_appointments[email] = set()

    user\_appointments[email].add((parsed\_date, selected\_doctor))

    appointment\_id = f"APPT{len(user\_appointments)}"

    user\_body = f"""Hello {name},

Your appointment with {selected\_doctor} ({suggested\_specialty}) has been successfully booked.

Appointment Date: {parsed\_date}

Appointment Time: {appointment\_time}

Appointment ID: {appointment\_id}

Regards,

Hospital Management

"""

    send\_email(email, "Appointment Confirmation", user\_body)

    print("Appointment booked successfully! Confirmation email sent.")

def reschedule\_appointment():

    email = input("Enter your email: ")

    if email not in user\_appointments or not user\_appointments[email]:

        print("No existing appointments found!")

        return

    print("Your current appointments:")

    appointment\_list = list(user\_appointments[email])

    for idx, (date, doctor) in enumerate(appointment\_list):

        print(f"{idx + 1}. {doctor} on {date}")

    choice = int(input("Select an appointment to reschedule: ")) - 1

    if choice < 0 or choice >= len(appointment\_list):

        print("Invalid selection!")

        return

    old\_date, doctor = appointment\_list[choice]

    # Get the original appointment time from stored data

    old\_time = None

    for time, doc in appointments.get(old\_date, {}).items():

        if doc == doctor:

            old\_time = time

            break

    if old\_time is None:

        print("Error: Original appointment time not found!")

        return

    new\_date = input("Enter new appointment date (YYYY-MM-DD or DD-MM-YYYY or MM/DD/YYYY or DD/MM/YYYY): ")

    parsed\_new\_date = parse\_date(new\_date)

    # Ensure new date is valid and not earlier than the old date

    while not parsed\_new\_date or parsed\_new\_date < old\_date:

        print("Invalid date. You cannot reschedule to an earlier date!")

        new\_date = input("Enter a valid new appointment date: ")

        parsed\_new\_date = parse\_date(new\_date)

    while True:

        new\_time = input("Enter new appointment time (HH:MM AM/PM): ")

        # \*\*Prevent rescheduling to an earlier time on the same date\*\*

        if parsed\_new\_date == old\_date:

            old\_time\_obj = datetime.strptime(old\_time, "%I:%M %p")

            new\_time\_obj = datetime.strptime(new\_time, "%I:%M %p")

            if new\_time\_obj < old\_time\_obj:

                print("You cannot reschedule to an earlier time on the same date! Please enter a valid time.")

                continue # Ask for time again

        # Ensure the new time slot is available

        if parsed\_new\_date in appointments and new\_time in appointments[parsed\_new\_date]:

            print("This time slot is already booked! Please enter another appointment time.")

            continue # Ask for time again

        break # Exit loop when valid time is entered

    # Remove the old appointment and add the new one

    appointments[old\_date].pop(old\_time, None)

    appointments.setdefault(parsed\_new\_date, {})[new\_time] = doctor

    user\_appointments[email].remove((old\_date, doctor))

    user\_appointments[email].add((parsed\_new\_date, doctor))

    print("Appointment rescheduled successfully!")

    # Send email for rescheduling

    reschedule\_body = f"""Hello,

Your appointment with {doctor} has been rescheduled.

New Appointment Date: {parsed\_new\_date}

New Appointment Time: {new\_time}

Regards,

Hospital Management

"""

    send\_email(email, "Appointment Rescheduled", reschedule\_body)

    print("Appointment rescheduled successfully! Confirmation email sent.")

def cancel\_appointment():

    email = input("Enter your email: ")

    if email not in user\_appointments or not user\_appointments[email]:

        print("No existing appointments found!")

        return

    print("Your current appointments:")

    appointment\_list = list(user\_appointments[email])

    for idx, (date, doctor) in enumerate(appointment\_list):

        print(f"{idx + 1}. {doctor} on {date}")

    choice = int(input("Select an appointment to cancel: ")) - 1

    if choice < 0 or choice >= len(appointment\_list):

        print("Invalid selection!")

        return

    cancel\_date, doctor = appointment\_list[choice]

    # Remove from appointments

    for time, doc in appointments.get(cancel\_date, {}).items():

        if doc == doctor:

            del appointments[cancel\_date][time]

            break

    # Remove from user appointments

    user\_appointments[email].remove((cancel\_date, doctor))

    # Send email confirmation

    cancel\_body = f"""Hello,

Your appointment with {doctor} on {cancel\_date} has been successfully canceled.

Regards,

Hospital Management

"""

    send\_email(email, "Appointment Cancellation", cancel\_body)

    print("Appointment canceled successfully! Confirmation email sent.")

def medical\_info():

    disease = input("Enter the disease name: ")

    today = datetime.today().strftime("%Y-%m-%d")

    simulated\_count = random.randint(100, 5000)

    print(f"As of {today}, approximately {simulated\_count} patients are affected by {disease} in India.")

while True:

    choice = input("\nSelect: Book (1), Reschedule (2), Cancel (3), Medical Info (4), Exit (5): ").lower()

    if choice in ["1", "book"]:

        book\_appointment()

    elif choice in ["2", "reschedule"]:

        reschedule\_appointment()

    elif choice in ["3", "cancel"]:

        cancel\_appointment()

    elif choice in ["4", "medical info"]:

        medical\_info()

    elif choice in ["5", "exit"]:

        break

    else:

        print("Invalid selection. Try again.")

|  |  |
| --- | --- |
|  |  |