# Code Explanation: Flask-Based Medical Chatbot and Hospital Recommendation System

## 1. Introduction

This Flask-based web application combines a medical chatbot and hospital recommendation system. It allows users to interact with a chatbot, receive health-related suggestions, and get hospital recommendations based on specific medical conditions and geographical filters like state and district.

## 2. Key Features

1. \*\*Medical Chatbot:\*\* Suggests actions, foods to take/avoid, medicines, and advises severity level based on symptoms entered by the user.  
2. \*\*Hospital Recommendation:\*\* Recommends hospitals for specific medical conditions with filters like state and district.  
3. \*\*User Feedback:\*\* Collects user feedback and stores it in an Excel file.  
4. \*\*Receipt Generation:\*\* Generates a downloadable Word document summarizing chatbot responses and hospital recommendations.

## 3. Code Structure

### 3.1 File Paths

The script uses file paths to load Excel and CSV files for user details, medical advice, and hospital information. Paths like `D:\my workspace\...` indicate where data files are stored.

### 3.2 Data Loading

The `try-except` block loads:  
1. \*\*Medical Data:\*\* An Excel file containing symptom-based medical suggestions.  
2. \*\*Hospital Data:\*\* Multiple CSV files containing hospital details for various states.

If any file fails to load, an empty DataFrame is created as a fallback to avoid application crashes.

### 3.3 Flask Routes

#### 3.3.1 `/` - Login Route

The login page collects user information such as username, age, gender, email, and phone number. These details are stored in `user\_details.xlsx` using the `save\_user\_details` function.

#### 3.3.2 `/app` - Main Application Route

This route handles the core functionality of the app. It processes two key inputs:  
1. \*\*Chatbot Input:\*\* User-provided symptoms are passed to the `medical\_chatbot` function, which returns medical suggestions.  
2. \*\*Hospital Recommendation:\*\* Users provide a medical condition, state, and district, which are filtered using the `recommend\_hospitals` function.

The results are displayed on the webpage and saved in the session for generating a downloadable receipt.

#### 3.3.3 `/feedback` - Feedback Route

Users can submit feedback after logging in. The `add\_feedback\_to\_excel` function appends the feedback to the user's record in `user\_details.xlsx`.

#### 3.3.4 `/download\_receipt` - Receipt Generation

This route generates a Word document summarizing user details, chatbot suggestions, and selected hospitals. The document is created using the `python-docx` library and saved as `receipt.docx`.

#### 3.3.5 `/get\_districts` - Dynamic District Retrieval

Based on the selected state, this route dynamically filters the hospital data to return a list of districts available for that state.

## 4. Core Functions

### 4.1 save\_user\_details(details)

This function saves user details to the Excel file `user\_details.xlsx`. If the file already exists, it appends the new user details.

### 4.2 add\_feedback\_to\_excel(username, feedback)

This function adds user feedback to the existing Excel file. It ensures feedback is stored under the corresponding user's record.

### 4.3 medical\_chatbot(user\_input)

This function processes user-provided symptoms and matches them with entries in the medical dataset. It returns:  
1. Action to take.  
2. Foods to eat/avoid.  
3. Medicine suggestions.  
4. Severity level, with a special note if severity is high.

### 4.4 recommend\_hospitals(medical\_condition, state\_filter, district\_filter)

This function filters the hospital dataset based on the medical condition and optional filters for state and district. It returns hospital details like name, address, contact number, rating, state, and district.

## 5. Libraries Used

1. \*\*Flask:\*\* For creating the web application.  
2. \*\*Pandas:\*\* For loading and manipulating data from Excel/CSV files.  
3. \*\*python-docx:\*\* For generating Word documents.  
4. \*\*Werkzeug:\*\* For secure file handling.  
5. \*\*OS:\*\* For file path and existence management.

## 6. Conclusion

This Flask application integrates chatbot functionality, hospital recommendation, and feedback collection in a user-friendly interface. The code ensures efficient data handling, dynamic filtering, and document generation.

## ****1. Imports and Setup****

from flask import Flask, request, render\_template, redirect, url\_for, send\_file, session, jsonify, flash

import pandas as pd

import os

from docx import Document

from werkzeug.utils import secure\_filename

### ****Explanation:****

* **Flask**: A lightweight Python web framework to build web applications.
* **request**: To handle form inputs and data sent by users (e.g., POST requests).
* **render\_template**: To render HTML templates (like login.html, index.html).
* **redirect and url\_for**: Redirect the user to specific routes or pages.
* **send\_file**: To send files (like receipts) for download.
* **session**: To temporarily store user-specific data across requests.
* **jsonify**: Converts Python data into JSON format for API responses.
* **flash**: To display messages to the user.
* **pandas (pd)**: A library for working with Excel and CSV data files.
* **os**: To work with the file system (e.g., checking if files exist).
* **docx**: To create and manipulate Word documents.
* **secure\_filename**: Safely handle file names when uploading or saving files.

## ****2. Application Initialization****

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

app.secret\_key = "your\_secret\_key"

* **Flask(name)**: Initializes the Flask application.
* **secret\_key**: Used to secure sessions and flash messages. This should be a secret string.

## ****3. File Paths****

LOGIN\_FILE\_PATH = "user\_details.xlsx"

MEDICAL\_FILE\_PATH = r'D:\my workspace\book 2.xlsx'

HOSPITAL\_FILE\_PATH = [

r"D:\my workspace\Arunachala\_Pradesh\_hospital\_data.csv",

...

r"D:\my workspace\West\_Bengal\_data.csv"

]

* **LOGIN\_FILE\_PATH**: Path to an Excel file storing user login and feedback details.
* **MEDICAL\_FILE\_PATH**: Path to an Excel file containing medical data for the chatbot.
* **HOSPITAL\_FILE\_PATH**: A list of CSV files, each containing hospital data for different states.

## ****4. Loading Datasets****

try:

medical\_data = pd.read\_excel(MEDICAL\_FILE\_PATH)

hospital\_data = pd.concat(

[pd.read\_csv(file, encoding='latin1') for file in HOSPITAL\_FILE\_PATH],

ignore\_index=True

)

hospital\_data.fillna('Unknown', inplace=True)

except Exception as e:

print(f"Error loading files: {e}")

medical\_data, hospital\_data = pd.DataFrame(), pd.DataFrame()

### ****Explanation:****

1. **medical\_data**: Reads the medical Excel file (book 2.xlsx) into a Pandas DataFrame.
2. **hospital\_data**: Combines all CSV hospital files into a single DataFrame using pd.concat.
3. **fillna('Unknown')**: Replaces any missing data (NaN) with the word "Unknown".
4. **Error handling**: If any file is missing or there’s an issue, an empty DataFrame is created.

## ****5. Helper Functions****

### ****a. Save User Details****

def save\_user\_details(details):

df = pd.DataFrame([details])

try:

if os.path.exists(LOGIN\_FILE\_PATH):

existing\_data = pd.read\_excel(LOGIN\_FILE\_PATH)

updated\_data = pd.concat([existing\_data, df], ignore\_index=True)

updated\_data.to\_excel(LOGIN\_FILE\_PATH, index=False)

else:

df.to\_excel(LOGIN\_FILE\_PATH, index=False)

except Exception as e:

print(f"Error saving user details: {e}")

* Adds user details to the user\_details.xlsx file.
* If the file already exists, it appends new data; otherwise, it creates a new file.

### ****b. Add Feedback****

def add\_feedback\_to\_excel(username, feedback):

try:

if os.path.exists(LOGIN\_FILE\_PATH):

df = pd.read\_excel(LOGIN\_FILE\_PATH)

if 'Feedback' not in df.columns:

df['Feedback'] = ''

df.loc[df['username'] == username, 'Feedback'] = feedback

df.to\_excel(LOGIN\_FILE\_PATH, index=False)

else:

raise FileNotFoundError("user\_details.xlsx not found!")

except Exception as e:

print(f"Error adding feedback: {e}")

* Updates the "Feedback" column for a specific user in the user\_details.xlsx file.

### ****c. Medical Chatbot****

def medical\_chatbot(user\_input):

for \_, row in medical\_data.iterrows():

if row['Symptom'].lower() in user\_input.lower():

response = [

f"Action: {row['Action']}",

f"Foods to Take: {row['Food\_to\_taken']}",

f"Foods to Avoid: {row['Food\_to\_avoid']}",

f"Medicine: {row['Medicine']}",

f"Severity: {row['Severity']}"

]

if row['Severity'].strip().lower() == 'high':

response.append("Note: Severity is High. Visit a doctor immediately.")

return response

return ["I'm sorry, I couldn't understand your problem."]

* Checks for a symptom mentioned by the user in the medical\_data file.
* If found, it returns advice, including food to take, food to avoid, medicine, and severity.

### ****d. Hospital Recommendation****

def recommend\_hospitals(medical\_condition, state\_filter=None, district\_filter=None):

hospital\_data['State'] = hospital\_data['State'].str.strip().str.lower().fillna('unknown')

hospital\_data['District'] = hospital\_data['District'].str.strip().str.lower().fillna('unknown')

hospital\_data['Medical Condition'] = hospital\_data['Medical Condition'].str.strip().str.lower()

filtered\_data = hospital\_data[hospital\_data['Medical Condition'].str.contains(medical\_condition, na=False)]

if state\_filter:

filtered\_data = filtered\_data[filtered\_data['State'] == state\_filter.strip().lower()]

if district\_filter:

filtered\_data = filtered\_data[filtered\_data['District'] == district\_filter.strip().lower()]

return filtered\_data[['Hospital\_Name', 'Address', 'Contact\_number', 'Rating', 'State', 'District']]

* Filters hospitals based on the user’s medical condition, state, and district.

## ****6. Routes****

### ****Login Page****

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

def login():

if request.method == "POST":

user\_details = {

"username": request.form.get("username"),

"age": request.form.get("age"),

"gender": request.form.get("gender"),

"email": request.form.get("email"),

"phone": request.form.get("phone"),

}

save\_user\_details(user\_details)

session.update(user\_details)

return redirect(url\_for("index"))

return render\_template("login.html")

* Collects user details via a form and saves them.

### ****Main App Page****

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

def index():

chatbot\_response = []

hospital\_results = []

states = sorted(hospital\_data['State'].dropna().unique().tolist())

if request.method == "POST":

user\_input = request.form.get("user\_input")

condition = request.form.get("condition")

state = request.form.get("state")

district = request.form.get("district")

if not user\_input or not condition:

flash("Please provide inputs for both Chatbot and Hospital Recommendation.")

chatbot\_response = medical\_chatbot(user\_input)

hospital\_results = recommend\_hospitals(condition, state, district).to\_dict(orient='records')

session['chatbot\_response'] = chatbot\_response

session['hospital\_results'] = hospital\_results

return render\_template("index.html", chatbot\_response=chatbot\_response, hospital\_results=hospital\_results, states=states)

* Displays the chatbot and hospital recommendation form.
* Processes inputs and shows results.

### ****Feedback and Receipt****

* /feedback: Saves user feedback.
* /download\_receipt: Generates a Word file containing the chatbot response and hospital recommendations.

## ****7. Running the App****

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

app.run(debug=True)

* Starts the Flask application in debug mode.