

A Self-Diagnosis Medical Chatbot Using Artificial Intelligence

1. Project Overview

The application consists of two main components:

- **LoginApp:** Handles user authentication, including login and registration.
- **DoctorChatbotApp:** The core chatbot application where users interact with a virtual doctor to input symptoms, receive follow-up questions, and generate prescriptions.

Key Features:

- **User Authentication:**
 - Users can register with a unique username and password.
 - Users can log in using their credentials.
 - Credentials are stored in an SQLite database.
- **Chatbot Interaction:**
 - Users interact with a virtual doctor through a chat interface.
 - The chatbot asks a series of diagnostic questions (**e.g., age group, symptoms, vitals**).
 - Users can input responses via text or speech (using speech-to-text).
- **Symptom Analysis:**
 - The chatbot analyzes **user-provided symptoms and vital signs**.
 - Supports image upload for skin condition analysis (**e.g., rash, eczema, psoriasis, acne**).
- **Prescription Generation:**
 - Generates prescriptions based on **symptoms, age group, and severity**.
 - Includes warnings for serious symptoms or abnormal vitals.
 - Prescriptions are stored in the database and can be exported to PDF.
- **Prescription History:**
 - Users can view, export, or delete past prescriptions.
- **Multilingual Support:**
 - Supports multiple languages for chatbot responses using the translate library.
- **Theme Toggle:**
 - Users can switch between light and dark themes for better usability.
- **Additional Features:**
 - Text-to-speech for chatbot responses.
 - Severity slider to indicate symptom severity (mild, moderate, severe).
 - PDF export for prescriptions.
 - Logging for debugging and error tracking.

Libraries Used:

- **tkinter:** For creating the GUI.
- **PIL (Pillow):** For image handling (e.g., skin image uploads).
- **translate:** For translating chatbot responses into different languages.

- **pyttsx3**: For text-to-speech functionality.
 - **speech_recognition**: For speech-to-text functionality.
 - **sqlite3**: For database management.
 - **reportlab**: For generating PDF prescriptions.
 - **logging**: For error logging.
 - **colorsys**: For color analysis in image processing.
 - Other standard libraries: **re**, **os**, **datetime**.
-

Summary

Each library plays a specific role in enabling the chatbot's functionality:

- **tkinter** builds the GUI.
- **PIL** handles skin image uploads and analysis.
- **translate** supports multilingual responses.
- **pyttsx3** and **speech_recognition** enable voice interaction.
- **sqlite3** manages data storage.
- **reportlab** generates PDFs.
- **logging** tracks errors.
- **colorsys** assists in image color analysis.
- **re**, **os**, and **datetime** provide utility functions for text parsing, file handling, and timestamps.

1. tkinter

- **Purpose:** Creates the graphical user interface (GUI) for the application.
- **Role in Project:**
 - Used to build the login/registration window (`LoginApp`) and the main chatbot interface (`DoctorChatbotApp`).
 - Provides widgets like buttons, labels, text boxes, and tabs for user interaction.
- **Example:**
 - In `LoginApp`, `tkinter` creates the login window with username/password fields and buttons:

```
python
self.username_label = tk.Label(root, text="Username", bg='#f0f0f0', font=('Arial',
self.username_entry = tk.Entry(root, font=('Arial', 12))
```
 - In `DoctorChatbotApp`, it sets up the chat interface with a notebook (tabs), chat log, and buttons:

```
python
self.notebook = ttk.Notebook(self.root)
self.chat_log = scrolledtext.ScrolledText(self.scrollable_frame, state='disabled',
```

2. PIL (Pillow)

- **Purpose:** Handles image processing and manipulation.
- **Role in Project:**
 - Used to load and analyze skin images uploaded by users for detecting conditions like rash, eczema, psoriasis, or acne.
- **Example:**
 - In the `upload_image` method, `PIL` opens the uploaded image:

python

... Copy

```
self.uploaded_image = Image.open(file_path)
```

- In `analyze_image`, the image is resized and converted to RGB for color analysis:

python

... Copy

```
img = image.convert('RGB').resize((100, 100))  
pixels = list(img.getdata())
```

3. translate

- **Purpose:** Translates text into different languages.
- **Role in Project:**
 - Enables the chatbot to communicate in multiple languages (e.g., Hindi, Spanish) by translating the doctor's messages.
- **Example:**
 - The `translate_text` method uses the `Translator` class to translate messages based on the selected language:

python

... Copy

```
translator = Translator(to_lang=self.language)  
return translator.translate(text)
```

- Used in `display_message` to translate questions like "Is the patient a child or an adult?" into the user's chosen language.

4. `pyttsx3`

- **Purpose:** Provides text-to-speech functionality.
- **Role in Project:**
 - Reads out the doctor's messages aloud for accessibility.
- **Example:**
 - In `text_to_speech`, `pyttsx3` converts the doctor's message to speech:

python

...  Copy

```
self.tts_engine = pyttsx3.init()
self.tts_engine.say(text)
self.tts_engine.runAndWait()
```

- Called in `display_message` to vocalize messages like "Please describe the patient's symptoms in detail."

5. `speech_recognition`

- **Purpose:** Converts speech to text.
- **Role in Project:**
 - Allows users to input responses via voice using the "Speak" button.
- **Example:**
 - In `speech_to_text`, it captures audio and converts it to text using Google's API:

python

...  Copy

```
recognizer = sr.Recognizer()
with mic as source:
    audio = recognizer.listen(source)
user_response = recognizer.recognize_google(audio)
```

- The recognized text (e.g., "I have a fever") is inserted into the entry box and processed.
-

6. `sqlite3`

- **Purpose:** Manages a lightweight SQLite database.
- **Role in Project:**
 - Stores user credentials, medical conditions, user profiles, and prescription history in `medical_data.db`.
- **Example:**
 - In `init_user_database`, creates the `users` table for storing usernames and passwords:

python

...  Copy

```
self.cursor.execute('''
    CREATE TABLE IF NOT EXISTS users (
        username TEXT PRIMARY KEY,
        password TEXT NOT NULL
    )
''')
```

- In `DoctorChatbotApp`, the `conditions` table stores treatments:

python

...  Copy

```
self.cursor.execute('''
    CREATE TABLE IF NOT EXISTS conditions (
        id INTEGER PRIMARY KEY,
        name TEXT, symptom TEXT, age_group TEXT, severity TEXT, treatment TEXT
    )
''')
```



7. `reportlab`

- **Purpose:** Generates PDF documents.
- **Role in Project:**
 - Exports prescriptions to PDF files for both the latest prescription and selected history entries.
- **Example:**
 - In `export_to_pdf`, creates a PDF with the prescription text:

python

...  Copy

```
c = canvas.Canvas(filename, pagesize=letter)
c.setFont("Helvetica", 12)
for line in self.current_prescription.split("\n"):
    c.drawString(50, y, line)
    y -= 15
c.save()
```

- Saves files like `prescription_newuser_20250429_143022.pdf`.

8. `logging`

- **Purpose:** Logs errors and debugging information to a file.
- **Role in Project:**
 - Records errors (e.g., database issues, speech recognition failures) in `chatbot_errors.log` for troubleshooting.
- **Example:**
 - Configured at the start to log messages with timestamps:

python

...  Copy

```
logging.basicConfig(filename='chatbot_errors.log', level=logging.DEBUG,
                    format='%(asctime)s - %(levelname)s - %(message)s')
```

- Logs database errors in `init_database`:

python

...  Copy

```
logging.error(f"Database initialization error: {e}")
```

9. `colorsys`

- **Purpose:** Converts between color systems (e.g., RGB to HSV).
- **Role in Project:**
 - Used in image analysis to detect skin conditions by analyzing pixel colors (e.g., for eczema detection).
- **Example:**
 - In `analyze_image`, checks for low saturation (dry/dull skin for eczema):

python

... Copy

```
elif any(colorsys.rgb_to_hsv(p[0]/255, p[1]/255, p[2]/255)[1] < 0.2 for p in pixel):  
    return "eczema"
```

10. Other Standard Libraries

a. `re` (Regular Expressions)

- **Purpose:** Parses and extracts patterns from text.
- **Role in Project:**
 - Extracts vital signs (e.g., temperature, heart rate) from user input and timestamps from selected prescriptions.
- **Example:**
 - In `parse_vitals`, extracts temperature from user input like "temperature 98.6°F":

python

... Copy

```
temp_match = re.search(r'temperature\s*(\d+\.\d*)\s*(°F|°C|degrees)', response)
```

- In `delete_selected_prescription`, extracts the timestamp from selected text:

python

... Copy

```
timestamp_match = re.search(r'\[(\d{4}-\d{2}-\d{1,2} \d{2}:\d{2}:\d{2})\]', selected_text)
```

b. `os`

- **Purpose:** Provides operating system-related functionality.
- **Role in Project:**
 - Used indirectly through other libraries (e.g., `reportlab` for file handling), but not explicitly in the code.

c. `datetime`

- **Purpose:** Handles date and time operations.
- **Role in Project:**
 - Generates timestamps for prescriptions and PDF filenames.
- **Example:**
 - In `generate_prescription`, creates a timestamp for storing prescriptions:

```
python                                                                    ... Copy

timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
self.cursor.execute("INSERT INTO prescriptions (username, prescription_text, times
                    (self.username, prescription, timestamp))
```

- Used in `export_to_pdf` for PDF filenames:

```
python                                                                    ... Copy

filename = f"prescription_{self.username}_{datetime.now().strftime('%Y%m%d_%H%M%S'}
```

2. File Structure and Setup

- **File Name:** `doctor.py`
- **Database:** `medical_data.db` (SQLite database to store user credentials, medical conditions, user profiles, and prescription history).
- **Log File:** `chatbot_errors.log` (for logging errors and debugging).

Dependencies:

To run the project, the following Python libraries must be installed:

```
pip install Pillow translate pytsx3 SpeechRecognition reportlab
```

3. Code Structure

The code is divided into three main parts:

1. **Imports and Setup:**
 - Imports all required libraries.
 - Configures logging to write errors to `chatbot_errors.log`.

2. **DoctorChatbotApp Class:**
 - Implements the main chatbot application.
 - Handles user interactions, symptom analysis, prescription generation, and history management.
 3. **LoginApp Class:**
 - Implements the login and registration interface.
 - Manages user authentication.
 4. **Main Execution:**
 - Starts the application with the login window.
-

4. Detailed Explanation of Components

4.1. DoctorChatbotApp Class

This class is the core of the application, handling all chatbot-related functionality after the user logs in.

4.1.1. Initialization (`__init__`)

- **Purpose:** Sets up the GUI and initializes the chatbot's state.
- **Key Steps:**
 - Configures the main window (root) to 80% of the screen size and sets the title as "Online Doctor Chatbot".
 - Initializes attributes:
 - **self.language:** Default language (**en for English**).
 - **self.tts_engine:** Text-to-speech engine using pyttsx3.
 - **self.patient_data:** Dictionary to store patient information (e.g., **age_group**, **symptoms**, **vitals**).
 - **self.diagnosis_state:** Tracks the current stage of diagnosis (e.g., **age_group**, **vitals**, **initial**).
 - **self.follow_up_questions:** List of follow-up questions based on symptoms.
 - **self.prescription_history:** Stores past prescriptions.
 - **self.uploaded_image:** Stores the uploaded skin image (if any).
 - **self.theme:** Tracks the current theme (light or dark).
 - Calls **self.init_database()** to set up the database.
 - Sets up the GUI using **ttk.Notebook** with two tabs: **"Chat"** and **"Prescription History"**.

4.1.2. Database Setup (`init_database`)

- **Purpose:** Initializes the SQLite database (**medical_data.db**) with necessary tables and sample data.
- **Tables:**
 - **conditions:**
 - Stores medical conditions and treatments.
 - Columns: **id**, **name**, **symptom**, **age_group**, **severity**, **treatment**, **description**, **severity_info**, **causes**, **prevention**.

- Sample data includes treatments for conditions like fever, cough, headache, diarrhea, rash, eczema, psoriasis, and acne for different age groups and severities.
- **user_profiles:**
 - Stores user-specific data.
 - Columns: username (primary key), age_group, allergies, history, lifestyle.
- **prescriptions:**
 - Stores generated prescriptions.
 - Columns: id (auto-incremented), username, prescription_text, timestamp.
- **Sample Data:**
 - Predefined treatments for various symptoms, age groups (child/adult), and severities (mild/moderate/severe).
 - Example: For a mild fever in an adult, the treatment is "Acetaminophen 500mg every 6 hours as needed (max 3g daily)."
- **Error Handling:**
 - Logs database errors to chatbot_errors.log.
 - Displays an error message if database initialization fails.

4.1.3. GUI Setup

- **Tabs:**
 - **Chat Tab:**
 - Contains the main chatbot interface.
 - Includes a canvas with a scrollbar for the chat log.
 - **Components:**
 - **Toggle Theme Button:** Allows switching between light and dark themes.
 - **Heading:** "ONLINE DOCTOR CHATBOT".
 - **Disclaimer:** Warns that the chatbot is not a substitute for professional medical advice.
 - **Chat Log:** Displays the conversation (read-only scrolledtext widget).
 - **Severity Slider:** Allows users to set symptom severity (1: mild, 2: moderate, 3: severe).
 - **Input Frame:**
 - **Entry Box:** For typing user responses.
 - **Send Button:** Submits the user's response.
 - **Speak Button:** Activates speech-to-text for voice input.
 - **Reset Chat Button:** Resets the chat to start over.
 - **Additional Buttons Frame:**
 - **Language Combobox:** Allows selecting a language for chatbot responses.
 - **Upload Skin Image Button:** Uploads an image for skin condition analysis.
 - **Generate Prescription Button:** Manually triggers prescription generation.
 - **Export Latest to PDF Button:** Exports the latest prescription to a PDF file.
 - **Prescription History Tab:**
 - Displays past prescriptions in a scrolledtext widget.
 - **Buttons:**
 - **Export Selected to PDF:** Exports the selected prescription to a PDF.

- **Delete:** Deletes the selected prescription from the database and history log.

4.1.4. Diagnostic Workflow

- **Diagnosis States:**
 - The chatbot follows a structured sequence of questions (**self.questions**):
 - **age_group:** Asks if the patient is a child or adult.
 - **vitals:** Requests vital signs (e.g., **temperature, heart rate**).
 - **initial:** Asks for a detailed description of symptoms.
 - **follow_up:** Asks symptom-specific follow-up questions (e.g., "Is the rash itchy?").
 - **duration:** Asks how long symptoms have been present.
 - **allergies:** Inquires about allergies or pre-existing conditions.
 - **history:** Asks if the patient has had similar symptoms before.
 - **lifestyle:** Requests details about diet, exercise, or travel.
 - **final:** Indicates the end of the question sequence, triggering prescription generation.
- **Follow-Up Questions:**
 - Generated based on symptoms using **self.follow_up_templates**.
 - **Example: For a fever, follow-up questions include "Is the fever accompanied by a rash?" and "Do you have chills or night sweats?"**
- **Serious Symptoms:**
 - Checks for serious symptoms (e.g., **chest pain, difficulty breathing**) listed in **self.serious_symptoms**.
 - If detected, the prescription includes an urgent warning to seek emergency care.

4.1.5. User Input Handling (send_response)

- **Purpose:** Processes user responses and advances the diagnosis state.
- **Steps:**
 - Retrieves the user's input from the entry box.
 - Displays the user's message in the chat log.
 - Based on the current **diagnosis_state**:
 - Validates and stores the response in **self.patient_data**.
 - **Example: For age_group, ensures the response is "child" or "adult".**
 - Generates follow-up questions if applicable (e.g., **after symptom input**).
 - Updates the severity from the slider (**mild, moderate, or severe**).
 - Advances to the next diagnosis state.
 - If the final state is reached, automatically calls **generate_prescription()**.

4.1.6. Speech-to-Text (speech_to_text)

- **Purpose:** Allows users to input responses via voice.
- **Implementation:**
 - Uses **speech_recognition** to capture audio from the microphone.
 - Converts speech to text using Google's speech recognition API.
 - Inserts the recognized text into the entry box and calls **send_response()**.
- **Error Handling:**
 - Handles cases where speech is not understood or the service is unavailable.

4.1.7. Text-to-Speech (text_to_speech)

- **Purpose:** Reads out the doctor's messages aloud.
- **Implementation:**
 - Uses **pyttsx3** to convert text to speech.
 - Called whenever the doctor sends a message to the chat log.

4.1.8. Image Upload and Analysis (upload_image and analyze_image)

- **Purpose:** Allows users to upload skin images for condition analysis.
- **Implementation:**
 - **Upload:**
 - Opens a file dialog to select an image (PNG, JPG, JPEG, GIF).
 - Loads the image using PIL and stores it in **self.uploaded_image**.
 - **Analysis:**
 - Resizes the image to 100x100 pixels for faster processing.
 - Analyzes pixel colors to detect skin conditions:
 - Redness (rash): High red channel values.
 - Whitish patches (psoriasis): High values in all RGB channels.
 - Yellowish pustules (acne): High red and green, low blue.
 - Dry/dull skin (eczema): Low saturation and dark pixels.
 - Returns the detected condition (e.g., "rash", "eczema") or None if no condition is detected.
 - **Follow-Up:**
 - Adds the detected condition to **self.patient_data["symptoms"]**.
 - Generates follow-up questions based on the detected condition.

4.1.9. Prescription Generation (generate_prescription)

- **Purpose:** Generates a prescription based on user input.
- **Steps:**
 - Validates that **age_group** is specified.
 - Initializes the prescription text with the patient's age group.
 - **Symptom Analysis:**
 - Checks for serious symptoms and adds an urgent warning if found.
 - Queries the conditions table for treatments matching the symptom, age group, and severity.
 - If no match is found, falls back to "mild" severity.
 - Includes treatment, description, severity info, causes, and prevention in the prescription.
 - **Vitals Analysis:**
 - Warns if temperature is high (e.g., >102°F for children, >103°F for adults).
 - Warns if heart rate is abnormal (<60 or >100 bpm).
 - **Patient Data:**
 - Includes allergies, symptom duration, medical history, and lifestyle factors.
 - **General Recommendations:**
 - Adds advice like verifying medications with a professional and monitoring symptoms.
 - **Storage:**
 - Saves the prescription to the prescriptions table with a timestamp.

- Updates the in-memory `prescription_history` and history log.
- **Display:**
 - Shows the prescription in a new window with a scrollable text area.

4.1.10. Prescription History Management

- **View:**
 - Displays all past prescriptions in the "Prescription History" tab with timestamps.
- **Export (`export_selected_to_pdf`):**
 - Allows exporting a selected prescription to a PDF file.
 - Extracts the timestamp from the selected text and retrieves the full prescription from the database.
- **Delete (`delete_selected_prescription`):**
 - Deletes the selected prescription from the database and updates the history log.

4.1.11. PDF Export (`export_to_pdf`)

- **Purpose:** Exports the latest prescription to a PDF file.
- **Implementation:**
 - Uses **reportlab** to create a PDF with the prescription text.
 - Saves the file with a filename like `prescription_{username}_{timestamp}.pdf`.
 - Displays a popup window mimicking an email client (Microsoft Outlook) to confirm the export.

4.1.12. Language Support

- **Purpose:** Allows the chatbot to communicate in multiple languages.
- **Implementation:**
 - Uses a **Combobox** to select from a list of supported languages (e.g., English, Hindi, Spanish).
 - Supports search functionality within the Combobox (`filter_languages`).
 - Translates doctor messages using the translate library (`translate_text`).
 - List of supported languages is stored in `self.languages` (e.g., 'Hindi': 'hi').

4.1.13. Theme Toggle (`toggle_theme`)

- **Purpose:** Switches between light and dark themes for better user experience.
- **Implementation:**
 - Toggles between light and dark themes by changing the background and foreground colors of all GUI elements.
 - Updates the "Toggle Theme" button text to reflect the current mode.

4.1.14. Reset Chat (`reset_chat`)

- **Purpose:** Resets the chat to start a new diagnosis session.
- **Implementation:**
 - Clears most patient data (except persistent fields like allergies, history, and lifestyle).
 - Resets the diagnosis state to **age_group**.
 - Clears the chat log and resets the severity slider.

4.2. LoginApp Class

This class handles user authentication before launching the chatbot.

4.2.1. Initialization (`__init__`)

- **Purpose:** Sets up the login/registration window.
- **GUI Components:**
 - **Username Entry:** Text field for the username.
 - **Password Entry:** Text field for the password (masked with *).
 - **Button Frame:**
 - **Login Button:** Triggers **check_login**.
 - **Register Button:** Triggers **register_user**.
- **Database:**
 - Calls **init_user_database()** to set up the users table.

4.2.2. User Database Setup (`init_user_database`)

- **Purpose:** Creates the users table in **medical_data.db**.
- **Table Structure:**
 - Columns: username (primary key), password.
- **Error Handling:**
 - Logs errors and displays a message if database initialization fails.

4.2.3. Login (`check_login`)

- **Purpose:** Verifies user credentials and launches the chatbot if valid.
- **Steps:**
 - Retrieves the username and password from the entry fields.
 - Queries the users table to check if the username exists and the password matches.
 - If valid, closes the login window and launches **main_app(username)**.
 - If invalid, displays an error message: **"Incorrect Username or Password"**.

4.2.4. Registration (`register_user`)

- **Purpose:** Allows new users to create an account.
- **Steps:**
 - Retrieves the username and password from the entry fields.
 - Checks if the username already exists in the users table.
 - If the username is unique, inserts the new user into the users table.
 - Displays a success message: "Registration successful! Please log in with your new credentials."
 - Clears the entry fields for a new login attempt.
 - If the username exists, displays an error: "Username already exists."

4.3. Main Execution

- **Purpose:** Starts the application.
- **Implementation:**

- Creates a **tk.Tk** instance for the login window.
- Initializes **LoginApp** to display the login/registration interface.
- Enters the **Tkinter main loop** to handle user interactions.

=====