HEALTHCARE CHATBOT SYSTEM

Bachelor of Technology

In

Computer Science and Engineering (Internet of Things)

Design & developed by

T. Abhinay Reddy	2111CS050068
D. Nikhil Reddy	2111CS050113
M. Nihanth Kumar	2111CS050112

Under the esteemed guidance

Mr. S. Rajasekhar Reddy

Assistant Professor

MALLA REDDY UNIVERSITY

Department of Computer Science and Engineering

Internet of Things

School of Engineering

MALLA REDDY UNIVERSITY

Maisammaguda, Dulapally, Hyderabad,500100

2021-2025

MALLA REDDY UNIVERSITY

(Telangana State Private Universities Act No.13 of 2020 and G.O.Ms.No.14, Higher Education (UE) Department)

Department of Computer Science and Engineering (Internet of Things)

CERTIFICATE

This is to certify that the application development project entitled "**HEALTHCARE CHATBOT SYSTEM**", submitted by **T. Abhinay Reddy (2111CS050068), D. Nikhil Reddy(2111CS050113), M. Nihanth Kumar(2111CS050112)** towards the partial fulfillment for the award of **Bachelor's Degree in Computer Science and Engineering** from the **Department of Internet of Things**, **Malla Reddy University**, Hyderabad, is a record of bonafide work done by him/ her. The results embodied in the work are not submitted to any other University or Institute for the award of any degree or diploma.

Internal Guide	**Head of the department**
Mr. S. Rajasekhar Reddy	**Dr. G.Anand Kumar**
(Assistant Professor)	**CSE(IOT)**

External Examine

DECLARATION

We hereby declare that the project report entitled "**HEALTHCARE CHATBOT SYSTEM**" has been carried out by us and this work has been submitted to the **Department of Computer Science and Engineering (Internet of Things), Malla Reddy University**, Hyderabad in partial fulfilment of the requirements for the award of degree of Bachelor of Technology. We further declare that this project work has not been submitted in full or part for the award of any other degree in any other educational institutions.

Place:

Date:

T. Abhinay Reddy	2111CS050068
D. Nikhil Reddy	2111CS050113
M. Nihanth Kumar	2111CS050112

ACKNOWLEDGEMENT

We extend our sincere gratitude to all those who have contributed to the completion of this project report. Firstly, we would like to extend our gratitude to **Dr. V. S. K Reddy**, **Vice Chancellor**, for his visionary leadership and unwavering commitment to academic excellence.

We would also like to express my deepest appreciation and extending gratitude to our project guide **Mr. S. Rajasekhar(Assistant Professor),** whose invaluable guidance, insightful feedback, and unwavering support have been instrumental throughout the course of this project for Successful outcome.

We extend our gratitude to our **PRC-convenor, Dr. G. Latha,** for giving valuable inputs and timely quality of our project through a critical review process.

We are also grateful to **Dr. G. Anand Kumar, Head of the Department of Internet of Things**, for providing with the necessary resources and facilities to carry out this project.

We would like to thank **Dr. Harikrishna Kamatham, Dean, School of Engineering**, for his encouragement and support throughout my academic pursuit.

T. Abhinay Reddy	2111CS050068
D. Nikhil Reddy	2111CS050113
M. Nihanth Kumar	2111CS050112

ABSTRACT

Today in the present era, the major challenges that India as a country is facing are to cater good quality and affordable healthcare services to its growing population and at the same time, they are not cost-efficient. This inaccessibility of healthcare facilities especially in rural areas and the difficulty in accessing means of transport causes patients to postpone their treatment, or option for medical facilities which is closer but at the same time are not cost-efficient and well-matched to their medical needs.

To tackle the above-mentioned problem, we have introduced an AI Healthcare Bot system. Our python-based system connects patients with the chatbot that will help them give the correct answers and precautions to their questions. It is developed with the aim to provide helpful information instantly, especially in times when every second is important. The system will also help the users find doctors, clinics, and hospitals nearby their location in emergencies.

Our python-based project comprises 2 modules: User and Admin. The user will need to register first to access the system. They can log in using their credentials after registering successfully. They can manage their profile and change the password if they want. The users can chat with the bot and resolve their health-related queries. They can also view doctors and their clinics near their location using Google places API. The users can also view hospitals nearby their location.

The admin can log in directly using their credentials to access the system. They can manage the question and answers and will require to train the model. They can also view the details of registered users.

We have created our own dataset to implement this system. Also, the CNN algorithm is used to develop this system. The accuracy of general disease risk prediction of CNN is higher as compared to other algorithms.

INDEX

CHAPTER-1
INTRODUCTION

1.1 Problem Definition & Description

In India, providing high-quality and affordable healthcare services remains a significant challenge, particularly for the growing population in rural areas. Limited accessibility to healthcare facilities and transportation difficulties result in delays in treatment or reliance on suboptimal medical services. This leads to inadequate health management, increased costs, and risks of undiagnosed or mistreated ailments.

The proposed solution is Healthcare ChatBot System, a Python-based chatbot designed to bridge this gap. By leveraging machine learning and natural language processing, this chatbot assists users in diagnosing illnesses, provides healthcare advice, and helps locate nearby medical facilities. The system empowers users with instant access to essential health information, reducing dependency on physical consultations and enabling timely responses in emergencies.

1.2 Objectives of the Project

To develop an AI-powered chatbot system that offers reliable, instant healthcare assistance, including disease diagnosis, health guidance, and the ability to locate nearby healthcare facilities, ensuring accessible and efficient medical support for users.

1.3 Scope of the project

- **Determining goals**

1. **Automated Health Assistance:**
   - Develop a chatbot capable of understanding and responding to user queries about symptoms and illnesses using natural language processing techniques.
2. **Enhanced Accessibility:**
   - Allow users to identify nearby hospitals, clinics, and healthcare providers using location-based services (Google Places API).
   - Ensure that healthcare resources are accessible, particularly for those in rural or underserved areas.
3. **Efficient User Interaction:**
   - Implement an interactive user interface that supports real-time text-based communication.
   - Provide features like user registration, login, and profile management.

CHAPTER - 2
SYSTEM ANALYSIS

2.1 Existing System

2.1.1 Background & Literature Survey

Srivastava and Singh (2020)[1] This study introduced an automated medical chatbot, **MediBot**, for providing virtual healthcare assistance. The system uses a machine learning approach to diagnose diseases based on user symptoms. The authors highlight the integration of deep learning techniques such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks for intent recognition and response generation. The chatbot is primarily designed to offer timely healthcare guidance and reduce dependency on physical consultations. However, they also noted challenges in dataset training and contextual understanding, suggesting further advancements in NLP tools.

Kandpal et al. (2020)[2] This research developed a contextual chatbot tailored for healthcare purposes using deep learning. The chatbot utilizes NLP to process natural language queries and provide personalized medical recommendations. The authors emphasize the chatbot's ability to handle multi-turn conversations and adapt responses based on prior inputs, ensuring a user-friendly interaction. Their findings indicate that integrating external APIs, such as location services, enhances the chatbot's functionality by connecting users to nearby medical facilities. They also identified limitations in scaling the chatbot for larger user bases.

Bharti et al. (2020)[3] The study presented **MedBot**, a conversational AI chatbot designed for tele-health applications. The authors focused on improving chatbot accuracy using Convolutional Neural Networks (CNNs) for symptom-disease prediction. The research highlighted the importance of real-time interaction, particularly during emergencies like the COVID-19 pandemic. The chatbot demonstrated significant potential in addressing basic healthcare needs, but the authors stressed the necessity of including multilingual support for wider adoption in diverse regions.

Sunny et al. (2018)[4] This study compared various machine learning algorithms for disease diagnosis systems, including decision trees, random forests, and support vector machines. It concluded that deep learning models such as RNNs provide superior accuracy due to their ability to learn sequential dependencies in user inputs. The research supports the use of structured datasets and data preprocessing techniques, such as lemmatization and tokenization, to enhance chatbot performance. It also advocates for extending chatbots' capabilities to include speech and image processing.

Rahman et al. (2019)[5] Rahman and colleagues developed a healthcare chatbot named **Disha**, focusing on symptom-based disease prediction using the Bangla language. The chatbot employed NLP techniques for text processing and integrated local healthcare datasets for disease prediction. The study highlighted the importance of tailoring healthcare solutions to regional needs and languages, addressing accessibility gaps. Challenges included limited training data and the need for more robust algorithms to handle diverse inputs effectively.

Mathew et al. (2019)[6]This paper proposed a chatbot system capable of disease prediction and treatment recommendation through machine learning. The chatbot employed supervised learning models trained on extensive datasets of symptoms and medical conditions. The authors demonstrated the efficiency of their model in handling large-scale data inputs while maintaining high accuracy. However, they noted that incorporating dynamic datasets and real-time updates could further enhance the system's reliability and utility.

2.1.2 Limitations of Existing System

Despite notable advancements, current systems suffer from several drawbacks:

1. **Limited Contextual Understanding:** Many chatbots fail to accurately interpret complex or nuanced health-related queries, leading to generic or incorrect advice.
2. **Dependence on Predefined Datasets:** Existing systems rely heavily on predefined datasets, limiting their ability to adapt to new or evolving user requirements.
3. **High Maintenance Costs:** Implementing and maintaining accurate healthcare chatbots requires significant investment in infrastructure, data training, and regular updates.
4. **Accessibility Issues:** The lack of integration with location-based services in rural and underserved areas reduces the utility of these systems.
5. **Human Dependency:** Traditional systems still rely on human intervention for complex decision-making, which reduces scalability and efficiency.

The proposed Healthcare ChatBot System aims to address these limitations by leveraging deep learning models, improving user interaction, and integrating advanced features for a seamless healthcare experience.

2.2 Proposed System

The Healthcare ChatBot system is designed as an innovative tool to bridge gaps in healthcare accessibility by providing real-time assistance for users in diagnosing health-related issues and offering location-based solutions. The system is divided into two modules, each designed for specific roles:

- **User Module:**

Registration and Login: Users must first register by providing personal details to access the system. Once registered, they can log in using their credentials.

Profile Management: Users can manage their profiles, including updating their information and changing passwords.

Chat with Bot: Users interact with a conversational chatbot to enter symptoms and receive relevant health-related advice. The bot also provides suggested medications or precautionary steps based on the provided symptoms.

Nearby Assistance: Using the Google Places API, the bot fetches and displays details about nearby doctors, clinics, and hospitals, aiding users in emergencies.

- **Admin Module:**

Data Management: Admins can add, edit, or delete question-and-answer pairs used by the chatbot to ensure accurate responses.

Model Training: Admins retrain the underlying machine learning model when new data or updates are introduced to improve accuracy.

User Management: The admin interface provides details about registered users for monitoring purposes.

- **System Functionality**

Chatbot Interaction: The chatbot leverages deep learning (CNN) to predict diseases based on symptoms entered by users. Its responses are tailored to provide instant help.

Real-Time Assistance: In critical situations, users can quickly locate nearby healthcare facilities, which is particularly beneficial for rural or remote areas.

Dataset Utilization: The bot is powered by a curated dataset comprising symptoms, diseases, and potential treatments, ensuring precise and context-aware responses.

2.2.1 Advantages of Proposed System

1. **Increased Health Awareness:**

Encourages individuals to consult the bot and identify potential health issues without delay, minimizing the tendency to postpone consultations.

2. **Accessibility:**

A viable solution for rural or underserved regions lacking sufficient healthcare infrastructure.

3. **Availability:**

Operates 24/7, eliminating the dependency on human intervention for basic consultations.

4. **Cost-Efficiency:**

Reduces operational costs by replacing traditional manpower with automated assistance.

5. **Enhanced User Experience:**

Quick responses ensure users find the system engaging and easy to use.

2.3 Software & Hardware Requirements

2.3.1 Software Requirements

1. **Programming Language:** Python, chosen for its simplicity and support for advanced AI/ML libraries.
2. **Core Libraries:**

- TensorFlow and Keras: For building and training the convolutional neural network (CNN) model.
- NLTK (Natural Language Toolkit): Enables the chatbot to process user inputs and provide accurate responses.
- Pickle: Used for serializing trained models and data for easy reuse.
- Google Places API: Retrieves location-based details of doctors, clinics, and hospitals.

3. **Development Tools**: Any Python-compatible IDE, such as PyCharm or Jupyter Notebook.

2.3.2 Hardware Requirements

1. **Processor:** Intel i5 or higher for efficient processing of deep learning tasks.
2. **RAM:** Minimum of 4 GB to handle model training and response generation.
3. **Operating System:** Windows 7 or above to ensure compatibility with the tools.

2.4 Feasibility Study

2.4.1 Technical Feasibility

The technical feasibility of the AI Health Care Bot system lies in its robust design and the use of cutting-edge technologies to achieve its goals. At the heart of the system is a convolutional neural network (CNN), a proven deep learning architecture known for its accuracy in pattern recognition and prediction tasks. The integration of tools like TensorFlow and Keras allows for streamlined model training, while NLTK ensures effective natural language processing for user interactions. The use of Google Places API enhances the system's capabilities by enabling real-time location-based services, allowing users to find nearby healthcare facilities effortlessly. Additionally, the reliance on Python, an accessible and versatile programming language, ensures compatibility with a wide range of devices and environments. The system's reliance on widely-used tools and frameworks means that its implementation and maintenance are both practical and scalable, catering to varying user needs while remaining technically efficient.

2.4.2 Robustness & Reliability

The robustness and reliability of the system stem from its well-designed architecture, which addresses potential challenges in a healthcare context. The CNN model ensures accurate predictions by learning from a comprehensive dataset of symptoms, diseases, and treatments, making it suitable for diverse health queries. Reliability is further enhanced through regular updates to the model and question-answer database, which keeps the system relevant and accurate. To handle real-world operational challenges, the system incorporates data serialization using Pickle, ensuring that trained models and data persist across sessions without performance loss. Security and reliability are maintained by employing Python's error-handling mechanisms, ensuring seamless execution even in cases of unexpected user inputs. Furthermore, the system's ability to provide instant assistance ensures consistent user trust and satisfaction, which are critical for healthcare applications.

2.4.3 Economic Feasibility

The economic feasibility of the AI Health Care Bot is highly favorable due to its cost-effective design and operational efficiency. The project leverages open-source tools and libraries, such as TensorFlow, Keras, and NLTK, which eliminate licensing costs while offering state-of-the-art performance. Hardware requirements are minimal, with a standard configuration of 4 GB RAM and an Intel i5 processor being sufficient to run the system effectively. For deployment, cloud-based services like Google Places API come at a nominal cost, ensuring scalability without significant financial burdens. Additionally, the elimination of manpower for basic healthcare consultations reduces ongoing operational expenses. In rural or underserved areas, where healthcare infrastructure is limited, the system offers a low-cost alternative to traditional services, making it an economically viable solution for improving healthcare accessibility on a large scale.

CHAPTER - 3
ARCHITECTURAL DESIGN

3.1 Module Design:

The system is modularly designed to enhance scalability, ease of development, and integration. The two main modules are:

3.1.1 User Module:

- Handles user interaction through a chatbot interface.
- Allows user registration, login, and profile management.
- Facilitates symptom entry and provides real-time responses about health conditions.
- Enables location-based searches for nearby doctors, clinics, and hospitals using APIs.

3.1.2 Admin Module:

- Manages the database of question-answer pairs and updates them regularly.
- Provides functionalities to retrain the machine learning model when required.
- Monitors and manages registered users and their interaction history.

3.2 Method & Algorithm design

3.2.1 Method

The AI Health Care Bot employs a combination of Natural Language Processing (NLP) and deep learning methods to process user inputs and generate accurate, meaningful responses. Key steps include:

- **Data Preprocessing:** Tokenization, stemming, and lemmatization are applied to user inputs to convert text into a machine-readable format.
- **Intent Classification:** The system identifies the user's intent based on their input, using a trained CNN.
- **Response Generation:** Based on the detected intent, the bot retrieves a predefined response or a prediction from the symptom-disease dataset.
- **Integration with External APIs:** Google Places API is used for retrieving location-specific information.

3.2.2 Algorithm

The chatbot uses a CNN-based algorithm for intent classification. Key steps include:

- **Input Layer:** User input is tokenized and represented as a numerical vector.
- **Hidden Layers:** Multiple dense layers process the data using ReLU activation functions to extract patterns.
- **Output Layer:** A softmax function predicts the intent with probabilities for each predefined category.

- **Response Selection:** The predicted intent is mapped to a response or further data lookup.

3.3 Project Architecture

3.3.1 Architecture Diagram

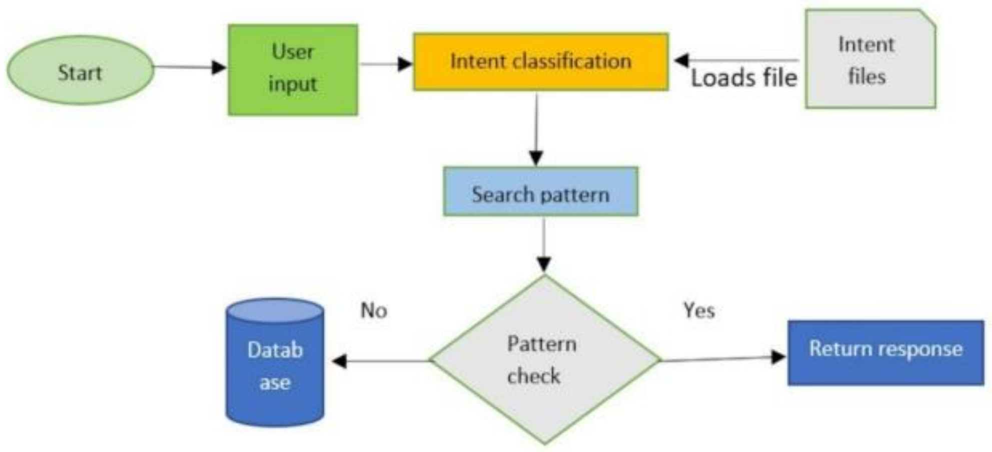


Fig 3.3.1 Architecture Diagram

This project architecture defines the interaction between the various system components, ensuring an organized and efficient workflow in the Healthcare ChatBot.

3.3.2 Data Flow Diagram

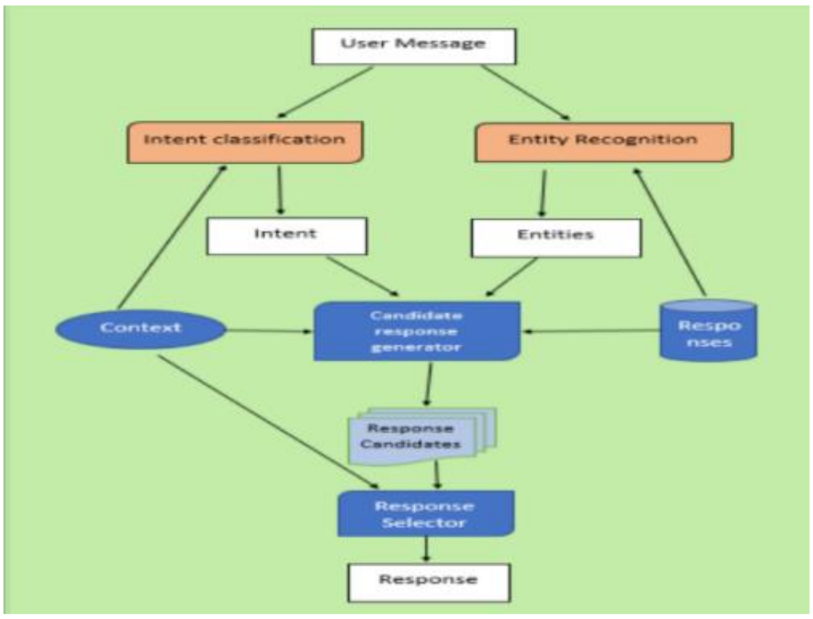


Fig 3.3.2 Data Flow Diagram

Level 0:

- User enters a query → System processes the query → System provides a response.

Level 1:

- User interacts with the bot.
- Query is processed for classification.
- Classified query leads to either:
    - A predefined response.
    - A lookup in the symptom-disease database.
- Results are returned to the user.

3.3.3 Class Diagram

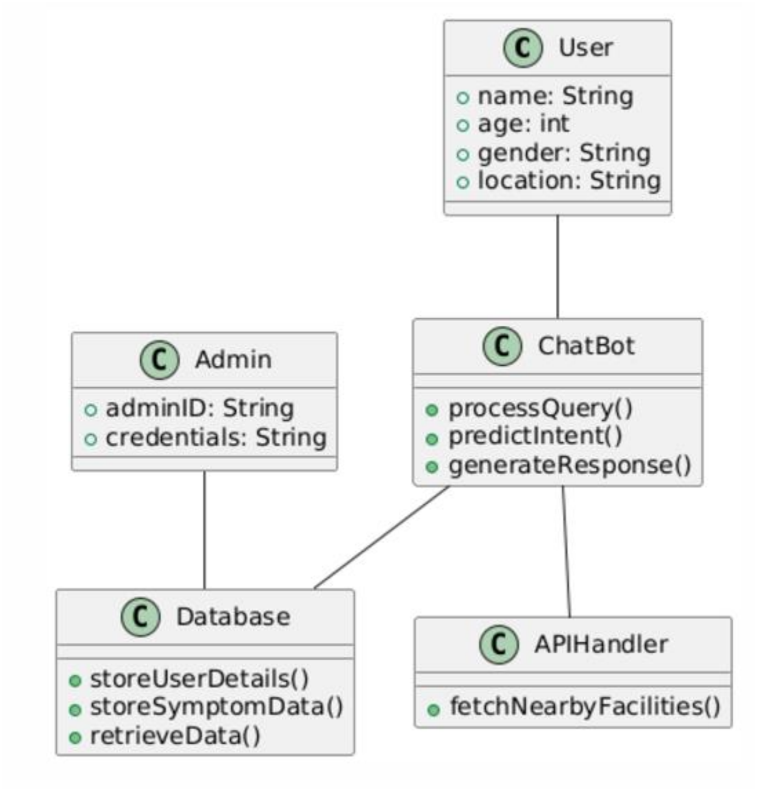


Fig 3.3.3 Class Diagram

The class diagram outlines the system's static structure:

- **Classes:**
    1. User: Attributes include name, age, gender, location.
    2. Admin: Attributes include adminID, credentials.

3. Chatbot: Methods for preprocessing, intent classification, and response generation.
4. Database: Stores user details, symptom-disease mappings, and question-answer pairs.
5. APIHandler: Manages external API requests.

3.3.4 Use Case Diagram

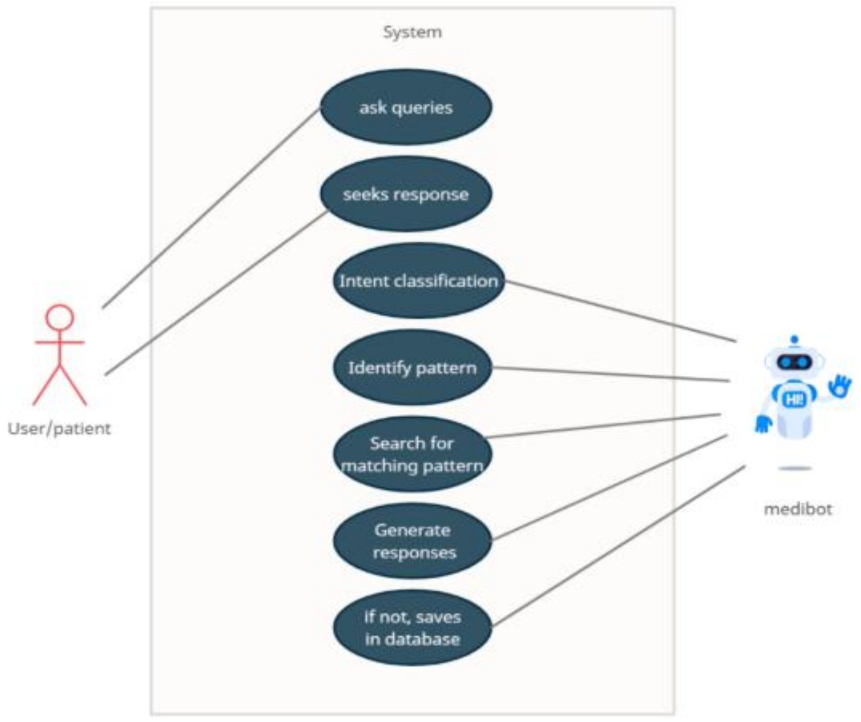


Fig3.3.4 Use Case Diagram

- Actors: User, Admin
- Use Cases:
    - User registers and logs in.
    - User enters symptoms or queries.
    - Bot provides a response or recommendation.
    - Admin updates question-answer data or retrains the model.
    - System retrieves and displays nearby healthcare facilities.

3.3.5 Sequence Diagram

- **Sequence for User Interaction:**

➢ User inputs a symptom/query.
➢ System preprocesses the query.
➢ System identifies the intent using the CNN model.
➢ System retrieves the response or additional information.
➢ Response is displayed to the user.

- **Sequence for Admin Action:**
  - Admin logs in.
  - Admin updates or retrains the model.
  - System processes updates and stores new data.

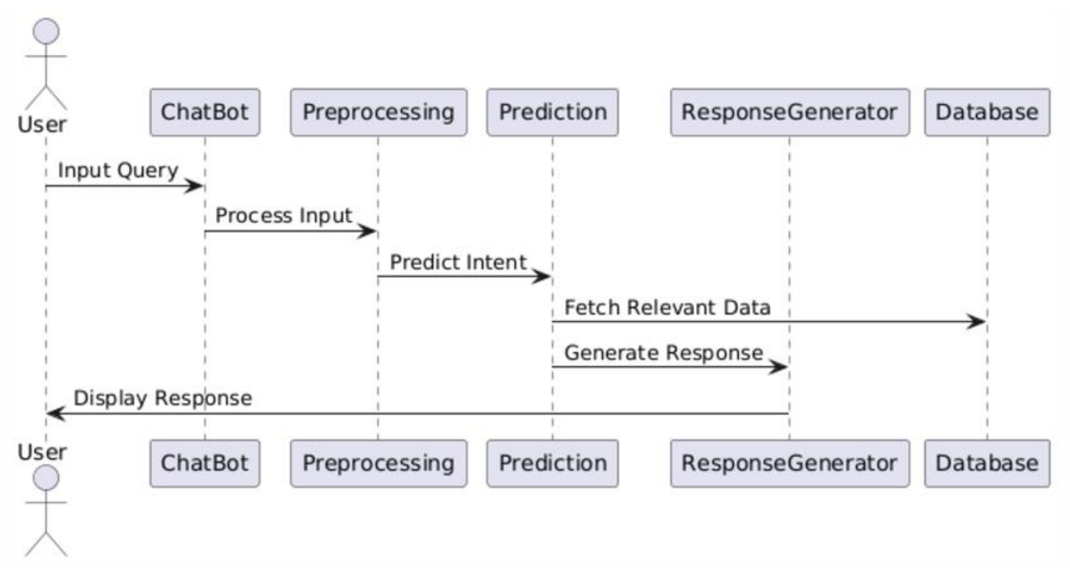


Fig 3.3.5 Sequence Diagram

3.3.6 Activity Diagram

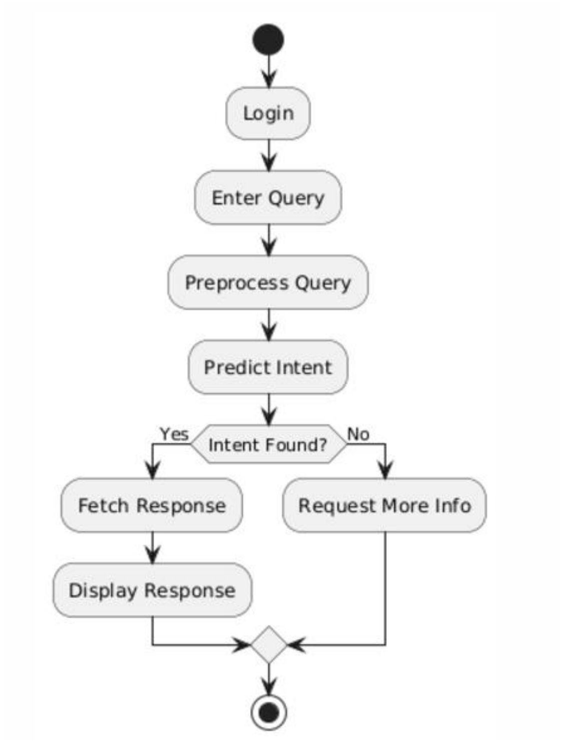


Fig3.3.6 Activity Diagram

- For User:

1. Start → Login → Enter Query → Preprocessing → Predict Intent → Generate Response → End.

- For Admin:

1. Start → Login → Access Dashboard → Update/Train Model → Save Changes → Logout → End.

CHAPTER-4
Implementation & Testing

4.1 Code Block

Source Code(app.py)

```
import random
from flask import jsonify
import secrets
from flask import Flask, render_template, flash, redirect, url_for, session, logging, request, session
from flask_sqlalchemy import SQLAlchemy
from collections.abc import Mapping

ALLOWED_EXTENSIONS = ['png', 'jpg', 'jpeg']
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
db = SQLAlchemy(app)
app.secret_key = "m4xpl0it"

def make_token():
    """
    Creates a cryptographically-secure, URL-safe string
    """
    return secrets.token_urlsafe(16)

class user(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80))
    email = db.Column(db.String(120))
    password = db.Column(db.String(80))

@app.route("/")
def index():
    return render_template("index.html")

userSession = {}

@app.route("/user")
def index_auth():
    my_id = make_token()
    userSession[my_id] = -1
    return render_template("index_auth.html",sessionId=my_id)

@app.route("/instruct")
def instruct():
    return render_template("instructions.html")

@app.route("/upload")
def bmi():
    return render_template("bmi.html")

@app.route("/diseases")
def diseases():
    return render_template("diseases.html")

@app.route('/pred_page')
def pred_page():
    pred = session.get('pred_label', None)
    f_name = session.get('filename', None)
    return render_template('pred.html', pred=pred, f_name=f_name)


@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        uname = request.form["uname"]
        passw = request.form["passw"]

        login = user.query.filter_by(username=uname, password=passw).first()
        if login is not None:
            return redirect(url_for("index_auth"))
    return render_template("login.html")

@app.route("/register", methods=["GET", "POST"])
def register():
    if request.method == "POST":
        uname = request.form['uname']
        mail = request.form['mail']
        passw = request.form['passw']

        register = user(username=uname, email=mail, password=passw)
        db.session.add(register)
        db.session.commit()

        return redirect(url_for("login"))
    return render_template("register.html")

import msgConstant as msgCons
import re

all_result = {
    'name':'',
    'age':0,
    'gender':'',
    'symptoms':[]
}

# Import Dependencies
# import gradio as gr
import pandas as pd
import numpy as np
from joblib import load
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

def predict_symptom(user_input, symptom_list):
    # Convert user input to lowercase and split into tokens
    user_input_tokens = user_input.lower().replace("_"," ").split()

    # Calculate cosine similarity between user input and each symptom
    similarity_scores = []
    for symptom in symptom_list:
        # Convert symptom to lowercase and split into tokens
        symptom_tokens = symptom.lower().replace("_"," ").split()

        # Create count vectors for user input and symptom
        count_vector = np.zeros((2, len(set(user_input_tokens + symptom_tokens))))
        for i, token in enumerate(set(user_input_tokens + symptom_tokens)):
            count_vector[0][i] = user_input_tokens.count(token)
            count_vector[1][i] = symptom_tokens.count(token)

        # Calculate cosine similarity between count vectors
        similarity = cosine_similarity(count_vector)[0][1]
        similarity_scores.append(similarity)

    # Return symptom with highest similarity score
    max_score_index = np.argmax(similarity_scores)
    return symptom_list[max_score_index]



import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Load the dataset into a pandas dataframe
df = pd.read_excel(r'E:\4.1 Application Development\Code\flask\dataset.xlsx')
```

```
# Get all unique symptoms
symptoms = set()
for s in df['Symptoms']:
    for symptom in s.split(','):
        symptoms.add(symptom.strip())


def predict_disease_from_symptom(symptom_list):

    user_symptoms = symptom_list
    # Vectorize symptoms using CountVectorizer
    vectorizer = CountVectorizer()
    X = vectorizer.fit_transform(df['Symptoms'])
    user_X = vectorizer.transform([', '.join(user_symptoms)])

    # Compute cosine similarity between user symptoms and dataset symptoms
    similarity_scores = cosine_similarity(X, user_X)

    # Find the most similar disease(s)
    max_score = similarity_scores.max()
    max_indices = similarity_scores.argmax(axis=0)
    diseases = set()
    for i in max_indices:
        if similarity_scores[i] == max_score:
            diseases.add(df.iloc[i]['Disease'])

    # Output results
    if len(diseases) == 0:
        return "<b>No matching diseases found</b>",""
    elif len(diseases) == 1:
        print("The most likely disease is:", list(diseases)[0])
        disease_details = getDiseaseInfo(list(diseases)[0])
        return f"<b>{list(diseases)[0]}</b><br>{disease_details}",list(diseases)[0]
    else:
        return "The most likely diseases are<br><b>"+ ', '.join(list(diseases))+"</b>",""



    symptoms = {'itching': 0, 'skin_rash': 0, 'nodal_skin_eruptions': 0, 'continuous_sneezing': 0,
            'shivering': 0, 'chills': 0, 'joint_pain': 0, 'stomach_pain': 0, 'acidity': 0, 'ulcers_on_tongue': 0,
            'muscle_wasting': 0, 'vomiting': 0, 'burning_micturition': 0, 'spotting_ urination': 0, 'fatigue': 0,
            'weight_gain': 0, 'anxiety': 0, 'cold_hands_and_feets': 0, 'mood_swings': 0, 'weight_loss': 0,
            'restlessness': 0, 'lethargy': 0, 'patches_in_throat': 0, 'irregular_sugar_level': 0, 'cough': 0,
            'high_fever': 0, 'sunken_eyes': 0, 'breathlessness': 0, 'sweating': 0, 'dehydration': 0,
            'indigestion': 0, 'headache': 0, 'yellowish_skin': 0, 'dark_urine': 0, 'nausea': 0, 'loss_of_appetite': 0,
            'pain_behind_the_eyes': 0, 'back_pain': 0, 'constipation': 0, 'abdominal_pain': 0, 'diarrhoea': 0, 'mild_fever': 0,
            'yellow_urine': 0, 'yellowing_of_eyes': 0, 'acute_liver_failure': 0, 'fluid_overload': 0, 'swelling_of_stomach': 0,
            'swelled_lymph_nodes': 0, 'malaise': 0, 'blurred_and_distorted_vision': 0, 'phlegm': 0, 'throat_irritation': 0,
            'redness_of_eyes': 0, 'sinus_pressure': 0, 'runny_nose': 0, 'congestion': 0, 'chest_pain': 0, 'weakness_in_limbs': 0,
            'fast_heart_rate': 0, 'pain_during_bowel_movements': 0, 'pain_in_anal_region': 0, 'bloody_stool': 0,
            'irritation_in_anus': 0, 'neck_pain': 0, 'dizziness': 0, 'cramps': 0, 'bruising': 0, 'obesity': 0, 'swollen_legs': 0,
            'swollen_blood_vessels': 0, 'puffy_face_and_eyes': 0, 'enlarged_thyroid': 0, 'brittle_nails': 0, 'swollen_extremeties': 0,
            'excessive_hunger': 0, 'extra_marital_contacts': 0, 'drying_and_tingling_lips': 0, 'slurred_speech': 0,
            'knee_pain': 0, 'hip_joint_pain': 0, 'muscle_weakness': 0, 'stiff_neck': 0, 'swelling_joints': 0, 'movement_stiffness': 0,
            'spinning_movements': 0, 'loss_of_balance': 0, 'unsteadiness': 0, 'weakness_of_one_body_side': 0, 'loss_of_smell': 0,
            'bladder_discomfort': 0, 'foul_smell_of urine': 0, 'continuous_feel_of_urine': 0, 'passage_of_gases': 0, 'internal_itching': 0,
            'toxic_look_(typhos)': 0, 'depression': 0, 'irritability': 0, 'muscle_pain': 0, 'altered_sensorium': 0,
            'red_spots_over_body': 0, 'belly_pain': 0, 'abnormal_menstruation': 0, 'dischromic _patches': 0, 'watering_from_eyes': 0,
            'increased_appetite': 0, 'polyuria': 0, 'family_history': 0, 'mucoid_sputum': 0, 'rusty_sputum': 0, 'lack_of_concentration': 0,
            'visual_disturbances': 0, 'receiving_blood_transfusion': 0, 'receiving_unsterile_injections': 0, 'coma': 0,
            'stomach_bleeding': 0, 'distention_of_abdomen': 0, 'history_of_alcohol_consumption': 0, 'fluid_overload.1': 0,
            'blood_in_sputum': 0, 'prominent_veins_on_calf': 0, 'palpitations': 0, 'painful_walking': 0, 'pus_filled_pimples': 0,
            'blackheads': 0, 'scurring': 0, 'skin_peeling': 0, 'silver_like_dusting': 0, 'small_dents_in_nails': 0, 'inflammatory_nails': 0,
            'blister': 0, 'red_sore_around_nose': 0, 'yellow_crust_ooze': 0}

    # Set value to 1 for corresponding symptoms

    for s in symptom_list:
        index = predict_symptom(s, list(symptoms.keys()))
        print('User Input: ',s," Index: ",index)
        symptoms[index] = 1
    # Put all data in a test dataset
    df_test = pd.DataFrame(columns=list(symptoms.keys()))
    df_test.loc[0] = np.array(list(symptoms.values()))
    print(df_test.head())
    # Load pre-trained model
    clf = load(str("model/random_forest.joblib"))
    result = clf.predict(df_test)

    disease_details = getDiseaseInfo(result[0])

    # Cleanup
    del df_test

    return f"<b>{result[0]}</b><br>{disease_details}",result[0]


import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Get all unique diseases
diseases = set(df['Disease'])

def get_symtoms(user_disease):
    # Vectorize diseases using CountVectorizer
    vectorizer = CountVectorizer()
    X = vectorizer.fit_transform(df['Disease'])
    user_X = vectorizer.transform([user_disease])

    # Compute cosine similarity between user disease and dataset diseases
    similarity_scores = cosine_similarity(X, user_X)

    # Find the most similar disease(s)
    max_score = similarity_scores.max()
    print(max_score)
    if max_score < 0.7:
        print("No matching diseases found")
        return False,"No matching diseases found"
    else:
        max_indices = similarity_scores.argmax(axis=0)
        symptoms = set()
        for i in max_indices:
            if similarity_scores[i] == max_score:
                symptoms.update(set(df.iloc[i]['Symptoms'].split(',')))
        # Output results
        print("The symptoms of", user_disease, "are:")
        for sym in symptoms:
            print(str(sym).capitalize())

        return True,symptoms

from duckduckgo_search import ddg

def getDiseaseInfo(keywords):
    results = ddg(keywords, region='wt-wt', safesearch='Off', time='y')
    print(results)
    return results
    #return results[0]['body']

@app.route('/ask',methods=['GET','POST'])
def chat_msg():

    user_message = request.args["message"].lower()
    sessionId = request.args["sessionId"]

    rand_num = random.randint(0,4)
    response = []
    if request.args["message"]=="undefined":

        response.append(msgCons.WELCOME_GREET[rand_num])
        response.append("What is your good name?")
        return jsonify({'status': 'OK', 'answer': response})
    else:

        currentState = userSession.get(sessionId)

        if currentState ==-1:
            response.append("Hi "+user_message+", To predict your disease based on symptopms, we need
some information about you. Please provide accordingly.")
            userSession[sessionId] = userSession.get(sessionId) +1
            all_result['name'] = user_message

        if currentState==0:
            username = all_result['name']
            response.append(username+", what is you age?")
            userSession[sessionId] = userSession.get(sessionId) +1

        if currentState==1:
            pattern = r'\d+'
            result = re.findall(pattern, user_message)
            if len(result)==0:
```

```
            response.append("Invalid input please provide valid age.")
        else:
            if float(result[0])<=0 or float(result[0])>=130:
                response.append("Invalid input please provide valid age.")
            else:
                all_result['age'] = float(result[0])
                username = all_result['name']
                response.append(username+", Choose Option ?")
                response.append("1. Predict Disease")
                response.append("2. Check Disease Symtoms")
                userSession[sessionId] = userSession.get(sessionId) +1

    if currentState==2:

        if '2' in user_message.lower() or 'check' in user_message.lower():
            username = all_result['name']
            response.append(username+", What's Disease Name?")
            userSession[sessionId] = 20
        else:

            username = all_result['name']
            response.append(username+", What symptoms are you experiencing?")
            response.append('<a href="/diseases" target="_blank">Symptoms List</a>')
            userSession[sessionId] = userSession.get(sessionId) +1

    if currentState==3:


        all_result['symptoms'].extend(user_message.split(","))
        username = all_result['name']
        response.append(username+", What kind of symptoms are you currently experiencing?")
        response.append("1. Check Disease")
        response.append('<a href="/diseases" target="_blank">Symptoms List</a>')
        userSession[sessionId] = userSession.get(sessionId) +1

    if currentState==4:

        if '1' in user_message or 'disease' in user_message:
            disease,type = predict_disease_from_symptom(all_result['symptoms'])
            response.append("<b>The following disease may be causing your discomfort</b>")
            response.append(disease)
            response.append(f'<a href="https://www.google.com/search?q={type} disease hospital near me" 
target="_blank">Search Near By Hospitals</a>')
            userSession[sessionId] = 10

        else:
            all_result['symptoms'].extend(user_message.split(","))
            username = all_result['name']
            response.append(username+", Could you describe the symptoms you're suffering from?")
            response.append("1. Check Disease")
            response.append('<a href="/diseases" target="_blank">Symptoms List</a>')
            userSession[sessionId] = userSession.get(sessionId) +1


        if currentState==5:
            if '1' in user_message or 'disease' in user_message:
                disease,type = predict_disease_from_symptom(all_result['symptoms'])
                response.append("<b>The following disease may be causing your discomfort</b>")
                response.append(disease)
                response.append(f'<a href="https://www.google.com/search?q={type} disease hospital near me"
target="_blank">Search Near By Hospitals</a>')

                userSession[sessionId] = 10

            else:

                all_result['symptoms'].extend(user_message.split(","))
                username = all_result['name']
                response.append(username+", What are the symptoms that you're currently dealing with?")
                response.append("1. Check Disease")
                response.append('<a href="/diseases" target="_blank">Symptoms List</a>')
                userSession[sessionId] = userSession.get(sessionId) +1

        if currentState==6:

            if '1' in user_message or 'disease' in user_message:
                disease,type = predict_disease_from_symptom(all_result['symptoms'])
                response.append("The following disease may be causing your discomfort")
                response.append(disease)
                response.append(f'<a href="https://www.google.com/search?q={type} disease hospital near me"
target="_blank">Search Near By Hospitals</a>')
                userSession[sessionId] = 10
            else:
                all_result['symptoms'].extend(user_message.split(","))
                username = all_result['name']
                response.append(username+", What symptoms have you been experiencing lately?")
                response.append("1. Check Disease")
                response.append('<a href="/diseases" target="_blank">Symptoms List</a>')
                userSession[sessionId] = userSession.get(sessionId) +1

        if currentState==7:
```

```
        if '1' in user_message or 'disease' in user_message:
            disease,type = predict_disease_from_symptom(all_result['symptoms'])
            response.append("<b>The following disease may be causing your discomfort</b>")
            response.append(disease)
            response.append(f'<a href="https://www.google.com/search?q={type} disease hospital near me"
target="_blank">Search Near By Hospitals</a>')
            userSession[sessionId] = 10
        else:
            all_result['symptoms'].extend(user_message.split(","))
            username = all_result['name']
           response.append(username+", What are the symptoms that you're currently dealing with?")
            response.append("1. Check Disease")
            response.append('<a href="/diseases" target="_blank">Symptoms List</a>')
            userSession[sessionId] = userSession.get(sessionId) +1

    if currentState==8:

        if '1' in user_message or 'disease' in user_message:
            disease,type = predict_disease_from_symptom(all_result['symptoms'])
            response.append("The following disease may be causing your discomfort")
            response.append(disease)
            response.append(f'<a href="https://www.google.com/search?q={type} disease hospital near me"
target="_blank">Search Near By Hospitals</a>')
            userSession[sessionId] = 10
        else:
            all_result['symptoms'].extend(user_message.split(","))
            username = all_result['name']
            response.append(username+", What symptoms have you been experiencing lately?")
            response.append("1. Check Disease")
            response.append('<a href="/diseases" target="_blank">Symptoms List</a>')
            userSession[sessionId] = userSession.get(sessionId) +1

    if currentState==10:
        response.append('<a href="/user" target="_blank">Predict Again</a>')


    if currentState==20:

        result,data = get_symtoms(user_message)
        if result:
            response.append(f"The symptoms of {user_message} are")
            for sym in data:
                response.append(sym.capitalize())

        else:response.append(data)
```

```
            userSession[sessionId] = 2
            response.append("")
            response.append("Choose Option ?")
            response.append("1. Predict Disease")
            response.append("2. Check Disease Symtoms")

        return jsonify({'status': 'OK', 'answer': response})
if __name__ == "__main__":
    with app.app_context():
        db.create_all()
    app.run(debug=False, port=3000)
```

4.3 Execution Flow

- ❖ **User Registration/Login:**
  - **New User:**
    - o The user registers by providing their name, age, gender, location, and password.
    - o The system stores the details in the database.
  - **Returning User:**
    - o The user logs in with their credentials.
    - o The system authenticates the user by verifying the credentials from the database.
- ❖ **Input Query:**
  - The user enters a query or symptoms in natural language through the chatbot interface.
- ❖ **Preprocessing:**
  - The system tokenizes the input text.
  - It performs lemmatization and removes stop words to clean the input.
  - Converts the input into a numerical vector for processing.
- ❖ **Intent Classification:**
  - The cleaned input is passed to the CNN model.
  - The model predicts the user's intent (e.g., symptom identification, general query, nearby facility search).
- ❖ **Response Generation:**
  - Based on the predicted intent:
    - o If it’s a symptom query, the system maps symptoms to potential diseases using the symptom-disease dataset.
    - o If it’s a general question, the system retrieves a predefined response from the database.
    - o If it’s a location-based query, the system calls the **Google Places API** to fetch nearby doctors, clinics, or hospitals.
  - The selected response is formatted and prepared for display.
- ❖ **Output Delivery:**
  - The system displays the response to the user through the chatbot interface.
  - In case of location-based queries, it provides a list of nearby healthcare facilities with details like address, distance, and ratings.
- ❖ **Admin Interaction:**

- Admins log in to manage data.
- They update the question-answer dataset, retrain the machine learning model, or monitor user activity logs.

❖ **Model Training (Admin-triggered):**

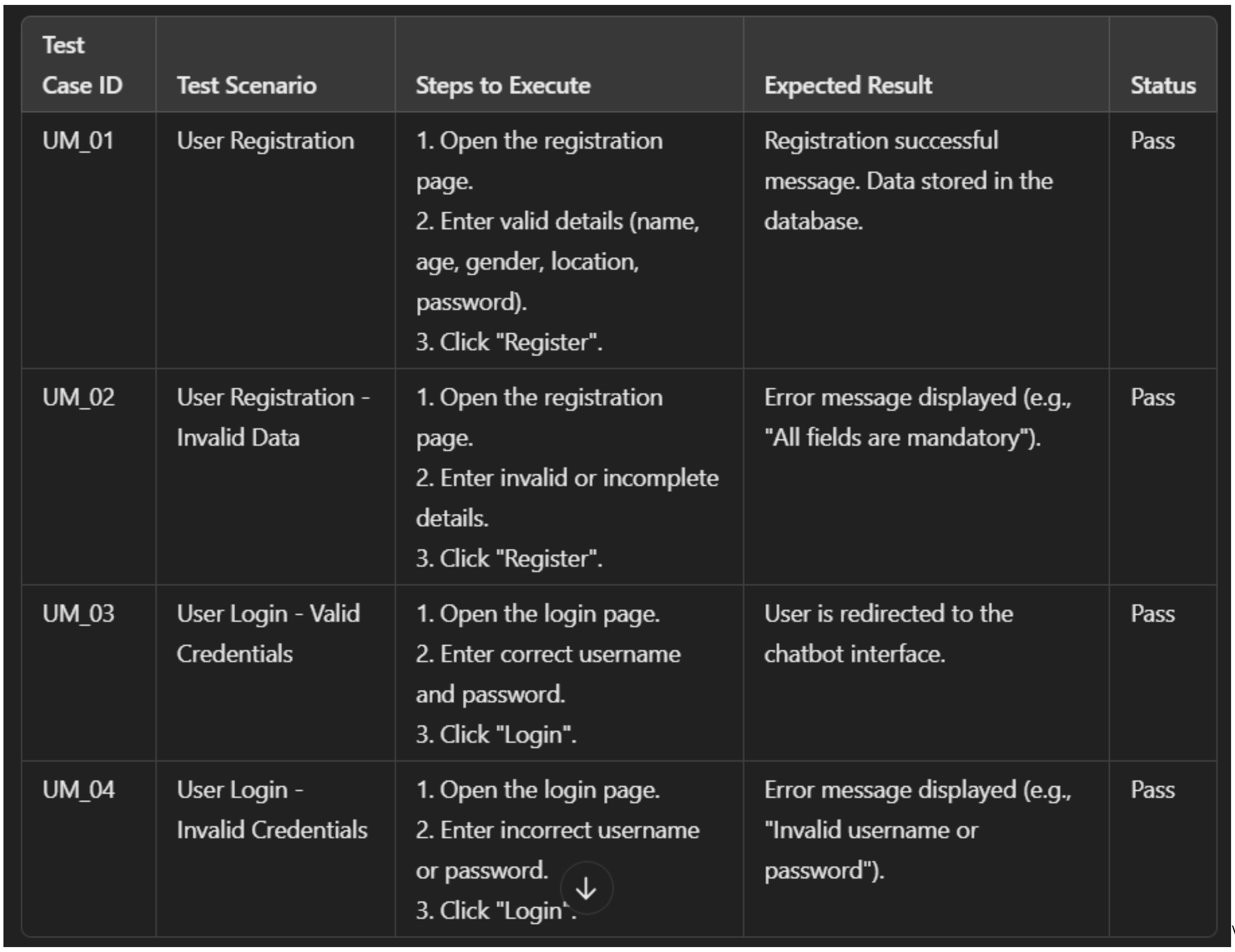
- Admin provides updated datasets.
- The CNN model is retrained using the new data for better accuracy.
- The updated model is deployed.

❖ **Session Termination:**

- The user logs out or closes the chatbot session.
- The system stores the interaction history for further analysis.

4.4 Testing

4.4.1 Test Case 1

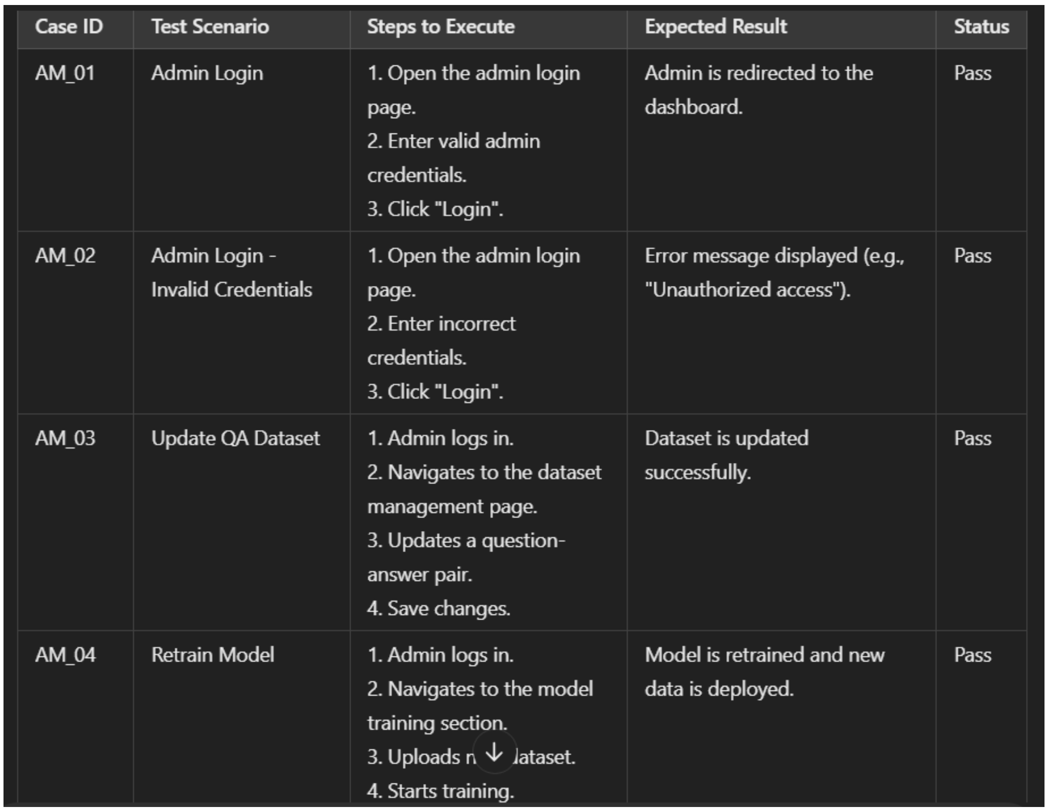


Test Case ID	Test Scenario	Steps to Execute	Expected Result	Status
UM_01	User Registration	1. Open the registration page. 2. Enter valid details (name, age, gender, location, password). 3. Click "Register".	Registration successful message. Data stored in the database.	Pass
UM_02	User Registration - Invalid Data	1. Open the registration page. 2. Enter invalid or incomplete details. 3. Click "Register".	Error message displayed (e.g., "All fields are mandatory").	Pass
UM_03	User Login - Valid Credentials	1. Open the login page. 2. Enter correct username and password. 3. Click "Login".	User is redirected to the chatbot interface.	Pass
UM_04	User Login - Invalid Credentials	1. Open the login page. 2. Enter incorrect username or password. 3. Click "Login".	Error message displayed (e.g., "Invalid username or password").	Pass

\

Fig 4.4.1 User Module Test Cases

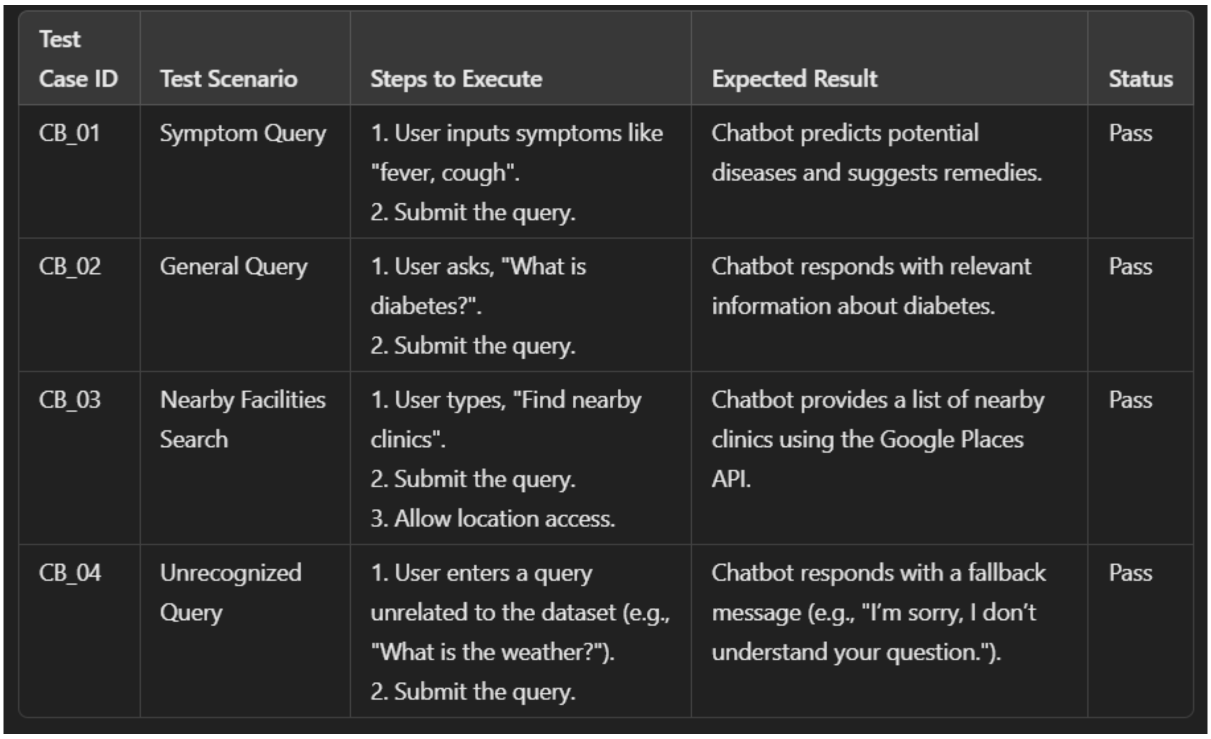
4.4.2 Test Case 2



Case ID	Test Scenario	Steps to Execute	Expected Result	Status
AM_01	Admin Login	1. Open the admin login page. 2. Enter valid admin credentials. 3. Click "Login".	Admin is redirected to the dashboard.	Pass
AM_02	Admin Login - Invalid Credentials	1. Open the admin login page. 2. Enter incorrect credentials. 3. Click "Login".	Error message displayed (e.g., "Unauthorized access").	Pass
AM_03	Update QA Dataset	1. Admin logs in. 2. Navigates to the dataset management page. 3. Updates a question-answer pair. 4. Save changes.	Dataset is updated successfully.	Pass
AM_04	Retrain Model	1. Admin logs in. 2. Navigates to the model training section. 3. Uploads n[illegible]dataset. 4. Starts training.	Model is retrained and new data is deployed.	Pass

Fig 4.4.2 Admin Module Test Cases

4.4.3 Test Case 3



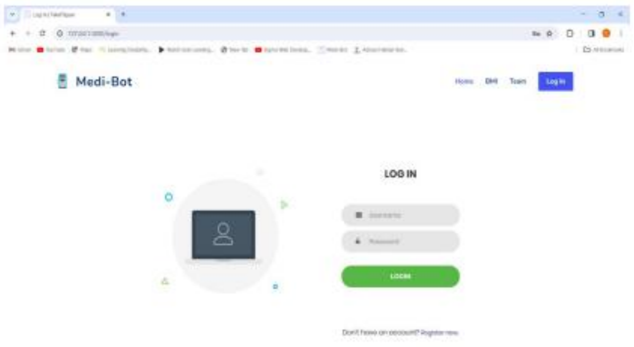
Test Case ID	Test Scenario	Steps to Execute	Expected Result	Status
CB_01	Symptom Query	1. User inputs symptoms like "fever, cough". 2. Submit the query.	Chatbot predicts potential diseases and suggests remedies.	Pass
CB_02	General Query	1. User asks, "What is diabetes?". 2. Submit the query.	Chatbot responds with relevant information about diabetes.	Pass
CB_03	Nearby Facilities Search	1. User types, "Find nearby clinics". 2. Submit the query. 3. Allow location access.	Chatbot provides a list of nearby clinics using the Google Places API.	Pass
CB_04	Unrecognized Query	1. User enters a query unrelated to the dataset (e.g., "What is the weather?"). 2. Submit the query.	Chatbot responds with a fallback message (e.g., "I'm sorry, I don't understand your question.").	Pass

Fig 4.4.3 Chatbot Functionalities Test Cases

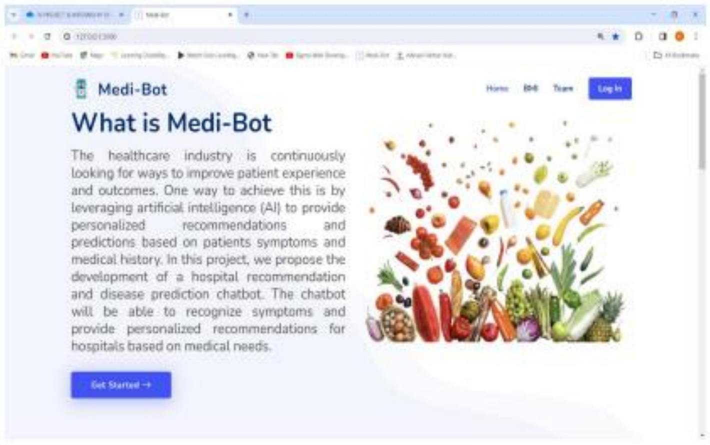
CHAPTER – 5
Results

5.1 Results Screens:

5.1.1 Screenshot 1



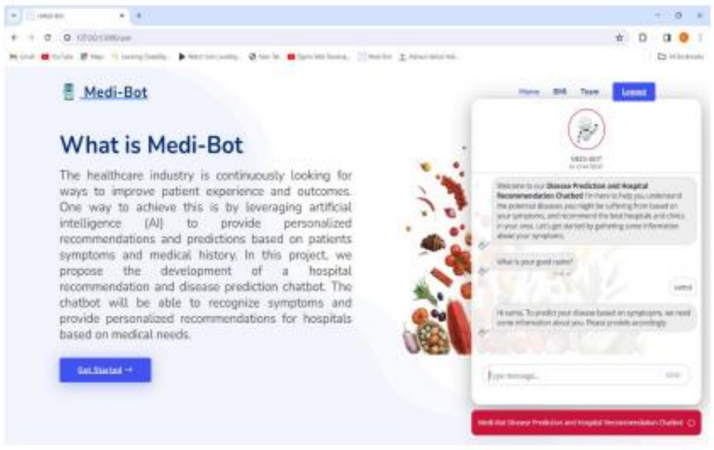
5.1.2 Screenshot 2



What is Medi-Bot

The healthcare industry is continuously looking for ways to improve patient experience and outcomes. One way to achieve this is by leveraging artificial intelligence (AI) to provide personalized recommendations and predictions based on patients symptoms and medical history. In this project, we propose the development of a hospital recommendation and disease prediction chatbot. The chatbot will be able to recognize symptoms and provide personalized recommendations for hospitals based on medical needs.

5.1.3 Screenshot 3



What is Medi-Bot

The healthcare industry is continuously looking for ways to improve patient experience and outcomes. One way to achieve this is by leveraging artificial intelligence (AI) to provide personalized recommendations and predictions based on patients symptoms and medical history. In this project, we propose the development of a hospital recommendation and disease prediction chatbot. The chatbot will be able to recognize symptoms and provide personalized recommendations for hospitals based on medical needs.

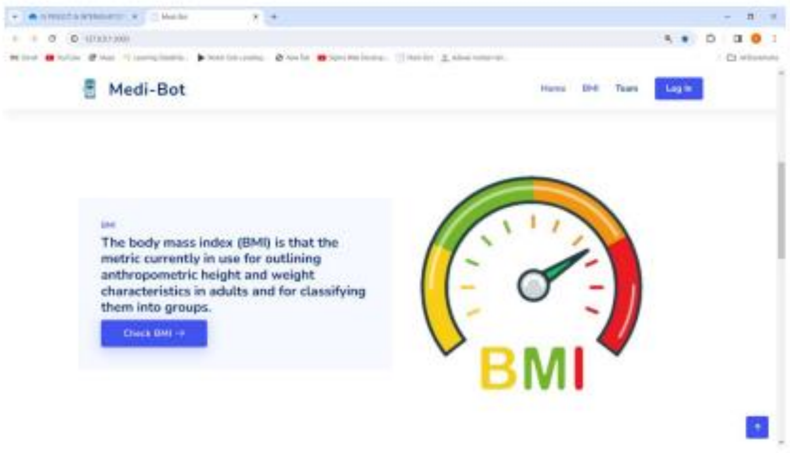
Welcome to our **Disease Prediction and Hospital Recommendation Chatbot** I'm here to help you understand the potential diseases you might be suffering from based on your symptoms, and recommend the best hospitals and clinics in your area. Let's get started by gathering some information about your symptoms.

What is your good name?

sumo

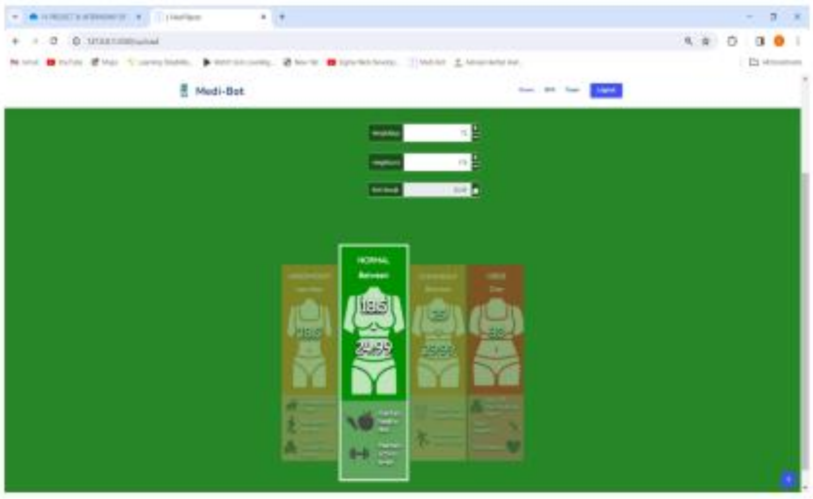
Hi sumo. To predict your disease based on symptoms, we need some information about you. Please provide accordingly.

5.1.4 Screenshot 4



The body mass index (BMI) is that the metric currently in use for outlining anthropometric height and weight characteristics in adults and for classifying them into groups.

5.1.5 Screenshot 5



CHAPTER – 6
CONCLUSION & FUTURE SCOPE

6.1 Conclusion

The Intent of this paper is to increase the awareness of health among the people. In current days, many people show their lazy behavior and don't consult a doctor during a time of illness so the implementation of a chatbot will help the people to diagnose the disease without consulting a doctor. The chatbot will act as a virtual doctor. The user will prescribe their symptoms of their illness and the chatbot will analyze the disease and suggest the necessary healthcare steps that need to be taken. In the datasets it includes information regarding diseases and health care steps.

6.2 Future Scope

Based on the system development and extensibility in future can also implement audio and face recognition to users benefits and also interact will doctors in case of a patient's emergency for treatment.

6.3 Reference

[1] Srivastava, P., & Singh, N (2020, February). Automatized medical chatbot (medibot). In 2020 International Conference on Power Electronics & IoT Applications in Renewable Energy and its Control (PARC) (pp. 351-354). IEEE.

[2] Kandpal, P., Jasnani, K., Raut, R., & Bhorge, S. (2020, July). Contextual Chatbot for healthcare purposes (using deep learning). In 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4) (pp.625-634). IEEE.

[3] Athota, L., Shukla, V. K., Pandey, N., & Rana, A. (2020, June). Chatbot for Healthcare System Using Artificial Intelligence. In 2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO) (pp. 619-622). IEEE.

[4] Karri, S. P. R., & Kumar, B. S. (2020, January). Deep learning techniques for implementation of chatbots. In 2020 International Conference on Computer Communication and Informatics (ICCCI) (pp. 1-5). IEEE.

[5] Bharti, U., Bajaj, D., Batra, H., Lalit, S., Lalit, S., & Gangwani, A. (2020, June). Medbot: Conversational artificial intelligence powered chatbot for delivering tele-healthafter covid-19. In 2020 5th International Conference on Communication and Electronics Systems (ICCES) (pp. 870-875). IEEE.

[6] Milla T Mutiwokuziva, Melody W Chanda, Prudence Kadebu, Addlight Mukwazvure, Tatenda T Gotora, "A Neural-network based Chat Bot" in ICCES,2017

[7] Gentner, T., Neitzel, T., Schulze, J., & Buettner, R. (2020, July). A Systematic literature review of medical chatbot research from a behavior change perspective. In 2020 IEEE 44th Annual Computers, Software, and Applications Conference

(COMPSAC) (pp. 735-740). IEEE.