ANUDIP FOUNDATION

Project Title

“Disease Prediction and Medical Recommendation System”

By

Name	Student id
Ajit swami	AF0481789

Under Guidance

Of

Rajshri Thete

ABSTRACT

The *Disease Prediction and Medical Recommendation System* is a web-based application developed using **Django** and powered by **Machine Learning** and **MySQL**. The primary goal of this project is to help users receive early disease predictions and suitable medical guidance based on their symptoms.

Users can input symptoms either by typing or through voice input. The system then processes the symptoms and matches them against a trained dataset to predict the most probable disease using fuzzy matching and database queries. Upon prediction, it provides a detailed report containing:

- **Disease description**
- **Precautionary measures**
- **Recommended medications**
- **Workout suggestions**

The application includes user authentication (signup/login/logout), session-based access, and secure form handling. Data is dynamically fetched from a **MySQL** database, ensuring real-time and personalized recommendations. This system can serve as a digital assistant for early diagnosis and personalized health suggestions.

ACKNOWLEDGEMENT

The project **"Disease Prediction and Medical Recommendation System"** is the Project work carried out by

Name	Enrollment No
Ajit Swami	**AF0481789**

Under the Guidance.

We are thankful to my project guide for guiding me to complete the Project.

His suggestions and valuable information regarding the formation of the Project Report have provided me a lot of help in completing the Project and its related topics.

We are also thankful to my family member and friends who were always there to provide support and moral boost up.

Content of the project

TITLE OF THE PROJECT

DISEASE PREDICTION AND MEDICAL RECOMMENDATION SYSTEM

Page No=> 1

INTRODUCTION OF THE PROJECT

The healthcare domain has significantly transformed the way medical services are delivered. One of the most impactful applications in the area of **disease prediction** and **medical recommendation systems**, which aim to provide timely, efficient, and accessible preliminary healthcare support to users.

This project, titled **"Disease Prediction and Medical Recommendation System"** focuses on developing a user-friendly web-based application that takes symptoms as input and predicts the most likely disease. Based on the predicted illness, the system also recommends relevant medicines and basic precautions to be followed. The primary goal is to assist users in identifying potential health conditions at an early stage and offer general medical advice without replacing the need for professional consultation.

The system uses machine learning concepts, identify patterns between input symptoms and diseases stored in a structured database. It is developed using **Python (Django Framework)** and integrates a **MySQL database** containing mappings of symptoms to diseases and medicine suggestions. The interface is built with **HTML, CSS, and JavaScript**, providing a clean and interactive platform for users to engage with the system.

This project is particularly useful in remote areas or situations where quick medical advice is needed and access to healthcare professionals may be limited. It also helps reduce the burden on medical institutions by enabling users to get initial recommendations digitally.

The system is designed to provide:

- Easy-to-understand output for non-technical users.
- A fast and responsive interface.
- Safe and informative advice to guide users toward professional help when necessary.

In essence, this research combines the power of AI with basic medical knowledge to create a practical tool that supports healthcare accessibility and awareness in a growing digital world.

OBJECTIVES

The primary objective of this research is to design and develop a user-friendly system capable of predicting diseases based on user-provided symptoms and recommending relevant medicines and basic health precautions. The system is intended to act as a digital health assistant that provides preliminary medical advice, especially in scenarios where immediate consultation with a doctor is not possible.

Specific Objectives:

1. **To Develop a Symptom-Based Disease Prediction Mode** Implement a system that uses symptom inputs to predict possible diseases through fuzzy matching techniques.
2. **To Recommend Suitable Medicines**
   Suggest relevant medicines for the predicted diseases using a predefined dataset, keeping safety and simplicity in mind.
3. **To Build an Interactive and User-Friendly Web Interface**
   Create a clean, responsive interface using HTML, CSS, and JavaScript for easy interaction, suitable for users with no technical background.
4. **To Integrate a Secure and Scalable Backend System**
   Use Python (Django framework) for backend operations and MySQL for structured data storage and retrieval.
5. **To Ensure Faster Response and Easy Accessibility**
   Optimize the system for quick disease predictions and ensure the application is lightweight and can be accessed via any browser.
6. **To Improve Health Awareness**
   Educate users by providing basic precautionary steps and advice relevant to their symptoms and conditions.
7. **To Simulate Real-Time Doctor-Like Conversations**
   Offer an experience that feels personal, helpful, and like talking to a medical assistant.
8. **To Minimize Dependency on Manual Medical Search**
   Eliminate the need for users to search on various websites or forums for symptom-related queries by giving instant results.

PROBLEM DEFINATION

Access to immediate and reliable healthcare advice continues to be a major challenge, especially in rural or underserved regions where medical professionals are few and far between. Many individuals delay seeking professional medical help due to long waiting times, lack of transportation, or simple unawareness of the seriousness of their symptoms. These delays can often result in the worsening of health conditions, which could have been managed effectively with early detection and guidance.

In this digital age, although various healthcare portals and apps exist, most either require detailed medical knowledge from users, offer generic responses, or fail to interpret user inputs accurately. Moreover, these systems often lack a smooth, conversational experience and do not provide tailored medicine suggestions based on predicted diseases.

The key problems identified are:

- **Limited Accessibility**: Inability to consult doctors immediately due to distance, cost, or infrastructure.
- **Delayed Diagnosis**: Many users ignore early symptoms due to the hassle of visiting a clinic or hospital.
- **Lack of Digital Guidance**: Existing apps often fail to offer understandable and symptom-specific recommendations.
- **Poor User Experience**: Many platforms are either too technical or non-intuitive for average users.
- **No Real-Time Interaction**: Absence of systems that can interpret symptoms in simple language and provide quick, useful advice.

This project aims to address these issues by building a web-based system that:

1. Accepts symptoms in a simple format from users.
2. Predicts the most probable disease using a fuzzy matching approach.
3. Recommends medicines and general precautions for the predicted disease.
4. Encourages users to seek professional medical assistance for confirmation and treatment.

By developing this system, we aim to provide a bridge between the user and medical guidance, especially for basic health concerns, through an AI-powered interface that is accurate, fast, and easy to use.

Preliminary Investigation

1. Purpose of the Project

The primary goal is to develop an intelligent web application that predicts diseases based on user-input symptoms and recommends necessary precautions, medications, workouts, and diet plans.
This project aims to assist users in identifying potential health issues early and taking preventive actions by providing instant, AI-supported medical suggestions.

2. Background

Due to the lack of immediate medical support and increasing health concerns, an early-stage disease prediction tool is crucial. By analysing symptoms, matching them with medical datasets, and utilizing Machine Learning concepts, the system predicts the most likely disease. Additionally, the platform offers medically researched recommendations to help users manage their health better.

3. Project Scope

- **User Management**: Sign-up, login, and session handling.
- **Symptom Input**: Manual typing or voice-based input.
- **Disease Prediction**: Matching symptoms using database queries and fuzzy matching techniques.
- **Medical Recommendations**: Displaying precautions, medications, diets, and workouts related to the disease.
- **Report Generation**: Users can download prediction reports in PDF format.
- **Database Integration**: All data fetched dynamically from a MySQL database.

4. Problems Identified

- Delay in getting preliminary medical advice.
- Users not aware of proper diet, workout, or precaution plans after identifying symptoms.
- Lack of tools for early disease prediction based on mild or early-stage symptoms.

- Manual search for symptoms and related diseases is time-consuming and confusing.

5. Proposed Solution

- Develop a Django web application that predicts diseases accurately based on symptoms.
- Integrate MySQL database for storing symptoms, diseases, precautions, medicines, diets, and workout data.
- Use a voice recognition feature for easier symptom input.
- Provide downloadable, professional-looking medical reports.
- User authentication system for personalized experience.

6. Feasibility Analysis

Criteria	Analysis
Technical Feasibility	Achievable using Django, MySQL, Bootstrap, and Python libraries.
Operational Feasibility	Easy-to-use UI, simple navigation, quick responses.
Economic Feasibility	No additional hardware/software cost beyond standard requirements.

7. Expected Benefits

- Early disease awareness for users.
- Time-saving and cost-effective.
- Easy access to preliminary health advice.

8. Constraints and Assumptions

- The system suggests potential diseases, not a confirmed medical diagnosis.
- The accuracy depends on the input symptoms and the data stored in the database.
- Requires basic internet connectivity to function.

FEASIBILITY STUDY

1. Introduction

The Disease Prediction and Medical Recommendation System is a web application that predicts possible diseases based on user-input symptoms and provides medical recommendations like medications, diet, workout routines, and precautions.
It uses Machine Learning concepts, a MySQL database, and a Django-based backend to deliver a user-friendly and helpful experience.

2. Purpose of the Feasibility Study

To evaluate whether this system can be successfully developed and deployed considering technical, operational, economic, and legal aspects.

3. Feasibility Types

3.1 Technical Feasibility

- **Technology Stack**: Django (Python), MySQL, Bootstrap 5, HTML/CSS, JavaScript (Speech Recognition API, html2pdf.js).
- **Database**: Stores user authentication data, disease information, symptoms, descriptions, and recommendations.
- **Speech Recognition**: Enables users to input symptoms by voice for accessibility.
- **Conclusion**:
  ✔ Technically feasible. Required software and frameworks are open-source and well-supported.

3.2 Operational Feasibility

- **Target Users**: General public seeking quick health advice, non-technical users, preliminary health assessments.
- **User Experience**: Simple, clean UI with speech-to-text and downloadable PDF reports.

- **Scalability**: Future addition of features like appointment booking, doctor consultation possible.

- **Conclusion**:
  ✔ Operable with basic training or instructions. Highly user-friendly.

3.3 Economic Feasibility

- **Cost**:
  - Development: Minimal, mostly time and effort.
  - Deployment: Can use free hosting initially (like Render, Vercel, or local servers).
- **Resources**:
  - Open-source technologies.
  - No special hardware requirements beyond standard server hosting.
- **Conclusion**:
  ✔ Economically feasible with low to negligible cost for a prototype.

3.4 Legal Feasibility

- **Data Privacy**:
  - No personal medical data is stored permanently.
  - Session-based operations ensure user privacy.
- **Compliance**:
  - Should mention that this system is **not a substitute for professional medical advice**.
- **Conclusion**:
  ✔ Legally feasible if proper disclaimers are used.

4. Risk Analysis

Risk	Likelihood	Impact	Mitigation
Incorrect Predictions	Medium	High	Improve ML model and database over time
User Misinterpretation	Medium	Medium	Clear disclaimers, advise to consult doctor
Server Downtime	Low	Medium	Reliable hosting and backups

Project Planning

1. Preliminary Investigation

- **Problem Identification**:
  Difficulty for patients to understand possible diseases based on symptoms without visiting a doctor immediately.

- **Proposed Solution**:
  A web-based AI system that predicts diseases based on user symptoms and provides recommendations like precautions, medications, workouts, and diet.

- **Feasibility Study**:
  - **Technical**: Possible using Django, MySQL, ML algorithms, and web technologies.
  - **Operational**: Easy to use for end-users (just entering symptoms).
  - **Economic**: Cost-effective since it uses open-source tools.

2. System Analysis

- **Existing Systems**:
  Mostly manual (doctor consultation), or complex health apps.

- **Requirements Gathering**:
  - User authentication (login/signup).
  - Symptom input through text or voice.
  - Disease prediction based on symptom match.
  - Medical recommendations (diet, medicine, workout, precautions).
  - PDF report generation for results.

- **Constraints**:
  - Only predictions — cannot replace actual medical diagnosis.
  - Database must be accurate and updated.

3. System Design

- **Architecture**:
    - Frontend: HTML, Bootstrap, JavaScript (Speech Recognition, PDF generation)
    - Backend: Django (Python)
    - Database: MySQL (storing symptoms, diseases, user info)
- **Flow**:
    1. User Login/Signup
    2. User enters symptoms.
    3. Symptoms matched with database.
    4. Disease predicted and recommendations fetched.
    5. Result displayed + option to download PDF.
- **Database Tables**:
    - users
    - symptom_severity
    - disease_training_data
    - disease_description
    - disease_precaution
    - disease_medicine
    - disease_diet
    - disease_workout

4. Coding

- **Backend**:
    - Django server routes
    - MySQL database connection
    - Predict disease logic using SQL queries.
- **Frontend**:
    - Form for entering symptoms.
    - Buttons for submitting and speaking symptoms.
    - Table for showing results.
    - Button for downloading results as PDF.
- **Speech Recognition**:
    - Using JavaScript's SpeechRecognition API.

5. Security

- **User Authentication**:
    - Login/Signup using session management.
    - User passwords securely stored (in future: consider password hashing).
- **Data Protection**:
    - Sessions to prevent unauthorized access.
    - Database access through parameterized queries to prevent SQL injection.

6. Testing

- **Unit Testing**:
    - Test database connections.
    - Test each route (login, signup, prediction).
- **Functional Testing**:
    - Test speech recognition.
    - Test PDF download feature.
- **User Testing**:
    - Check for user input errors (empty symptoms, invalid inputs).
    - Check if predictions and recommendations are meaningful.

7. Implementation

- **Deployment**:
    - Run Django server locally (localhost:5000).
    - In future: Deploy on platforms like Heroku, AWS, or PythonAnywhere.
- **Training Users**:
    - Simple user guide for login, entering symptoms, and downloading report.
- **Future Scope**:
    - Improve model accuracy using Machine Learning.
    - Add multi-language support.
    - Add symptom suggestions as user types.

Project Scheduling

Day	Task
Day 1	Setup, Environment, Project Planning
Day 2	Database schema design
Day 3	Fill database tables
Day 4–5	Backend (Signup, Login, Prediction APIs)
Day 6–7	Frontend UI (index.html, signup.html, login.html)
Day 8	Testing Features (PDF download, speech recognition)
Day 9	Bug Fixes, Final Tweaks
Day 10	Project Deployment or Submission

Software Requirement Specification

1. Project Title:

Disease Prediction and Medical Recommendation System

2. Purpose:

The system helps users **predict possible diseases** based on **their entered symptoms**. It also provides **recommendations** like **description of disease**, **precautions**, **medications**, **workout advice**, and **diet plans** to stay healthy.

3. Scope:

- Users can **sign up**, **log in**, and **log out**.
- After login, users can **enter symptoms manually** or **speak symptoms** using **voice input**.
- The system **predicts the most probable disease** from a database.
- The system **displays a full report** including:
    - Symptoms entered
    - Predicted disease
    - Disease description
    - Suggested precautions
    - Medications
    - Workout recommendations
    - Diet recommendations
- Users can **download the report as a PDF**.

4. Functional Requirements:

- **User Authentication**:
    - Sign up with username, email, and password.
    - Log in with email and password.
    - Logout.
- **Symptom Input**:
    - Manual input through text box.
    - Voice input using microphone button.
- **Disease Prediction**:
    - Match symptoms with database records.
    - Predict the most matching disease.
- **Recommendations**:
    - Display description, precautions, medications, workout, and diet for the predicted disease.
- **Report Download**:
    - Generate and download the prediction report as a PDF file.

5. Non-Functional Requirements:

- **Performance**:
    - The system should give results within a few seconds after entering symptoms.
- **Usability**:
    - Simple, attractive, and easy-to-use interface with mobile responsiveness.

- **Reliability**:
    - The system must be available and work correctly all the time.
- **Security**:
    - User passwords must be stored securely (recommended: use hashing in future).

6. Software and Tools Required:

- **Backend**: Python (Django Framework)
- **Frontend**: HTML, CSS (Bootstrap 5), JavaScript
- **Database**: MySQL
- **Libraries/Packages**:
    - Django
    - fuzzywuzzy
    - html2pdf.js
- **Development Tools**:
    - Visual Studio Code / PyCharm (for coding)
    - MySQL Workbench or phpMyAdmin (for database management)
- **Browser**:
    - Chrome, Firefox, or any modern browser.

7. System Architecture:

- **Client Side (Frontend)**:
  User enters symptoms -> Data sent to server -> Displays prediction and recommendations.

- **Server Side (Backend)**: Django app handles routes, processes input, queries database, and returns results.
- **Database**: Stores data like user details, symptoms, diseases, descriptions, diets, medicines, precautions, workouts.

8. Assumptions:

- Users have an active internet connection.
- Users allow microphone access (for voice input).
- Symptoms and disease data in the database are already available and properly maintained.

9. Constraints:

- Only diseases present in the database can be predicted.
- Voice input depends on browser compatibility (SpeechRecognition API).

10. Future Enhancements (Optional ideas):

- Use **machine learning models** to predict diseases more accurately.
- Add **password encryption** for better security.
- Allow users to **save past reports** in their account.
- Add a **chatbot** for better interaction.

Functional Requirements Document (FRD)

Project Title: Disease Prediction and Medical Recommendation System

1. Overview

The system allows users to predict potential diseases based on symptoms provided either through typing or voice input. It also recommends a description, precautions, medications, workout routines, and diet plans for the predicted disease.

The system uses a **Django** web server and **MySQL** as a backend database.

2. User Roles

- **User:** A person who signs up or logs into the system to predict diseases.
- **Admin (optional future expansion):** Not implemented yet but could manage database records.

3. Functional Requirements

3.1 User Authentication

- **Signup:**
  - Users can create an account by entering username, email, and password.
  - The system checks if the username or email already exists.
  - If already existing, an error message is displayed.
- **Login:**
  - Users can log in using email and password.
  - On successful login, they are redirected to the prediction page.
  - Invalid credentials show an error message.
- **Logout:**
  - Users can logout from their session.

3.2 Symptom Input

- **Symptom Typing:**
  - Users can type symptoms in an input field separated by commas.
- **Voice Input:**
  - Users can use a microphone button to speak symptoms.
  - Speech is converted into text and filled in the input box.

3.3 Disease Prediction

- **Predict Button:**
  - On submitting symptoms, the system matches them against the database.
  - It identifies the disease with the most symptom matches.
- **Disease Information Display:**
  - Symptoms entered by the user.
  - Predicted Disease Name.
  - Disease Description.
  - Recommended Precautions.
  - Suggested Medications.
  - Suggested Workouts.
  - Recommended Diet.

3.4 Report Generation

- **Download Report as PDF:**
  - After prediction, users can download the complete report as a PDF file using the **html2pdf.js** library.

3.5 Database Interactions

- **Fetch symptoms** from symptom_severity table.
- **Match symptoms** with disease_training_data to predict the disease.
- **Fetch description** from disease_description table.
- **Fetch diet recommendations** from disease_diet.
- **Fetch medication recommendations** from disease_medicine.
- **Fetch precautions** from disease_precaution.
- **Fetch workout routines** from disease_workout.

4. Non-Functional Requirements

- **Security:** Passwords are stored directly (plain text). (Should be encrypted in future improvements.)
- **Responsiveness:** Bootstrap is used to ensure mobile-friendliness.
- **Performance:** Efficient query execution using optimized SQL joins and limit clauses.
- **Reliability:** Handles missing or incomplete data gracefully (example: No description available).
- **User Interface:** Clean and modern UI using gradient backgrounds, blur effects, and animations.

5. Technical Stack

- **Frontend:**
  - HTML5, CSS3 (Bootstrap 5.3)
  - JavaScript (Speech Recognition API, html2pdf.js)
- **Backend:**
  - Python Django Framework
- **Database:**
  - MySQL

6. Error Handling

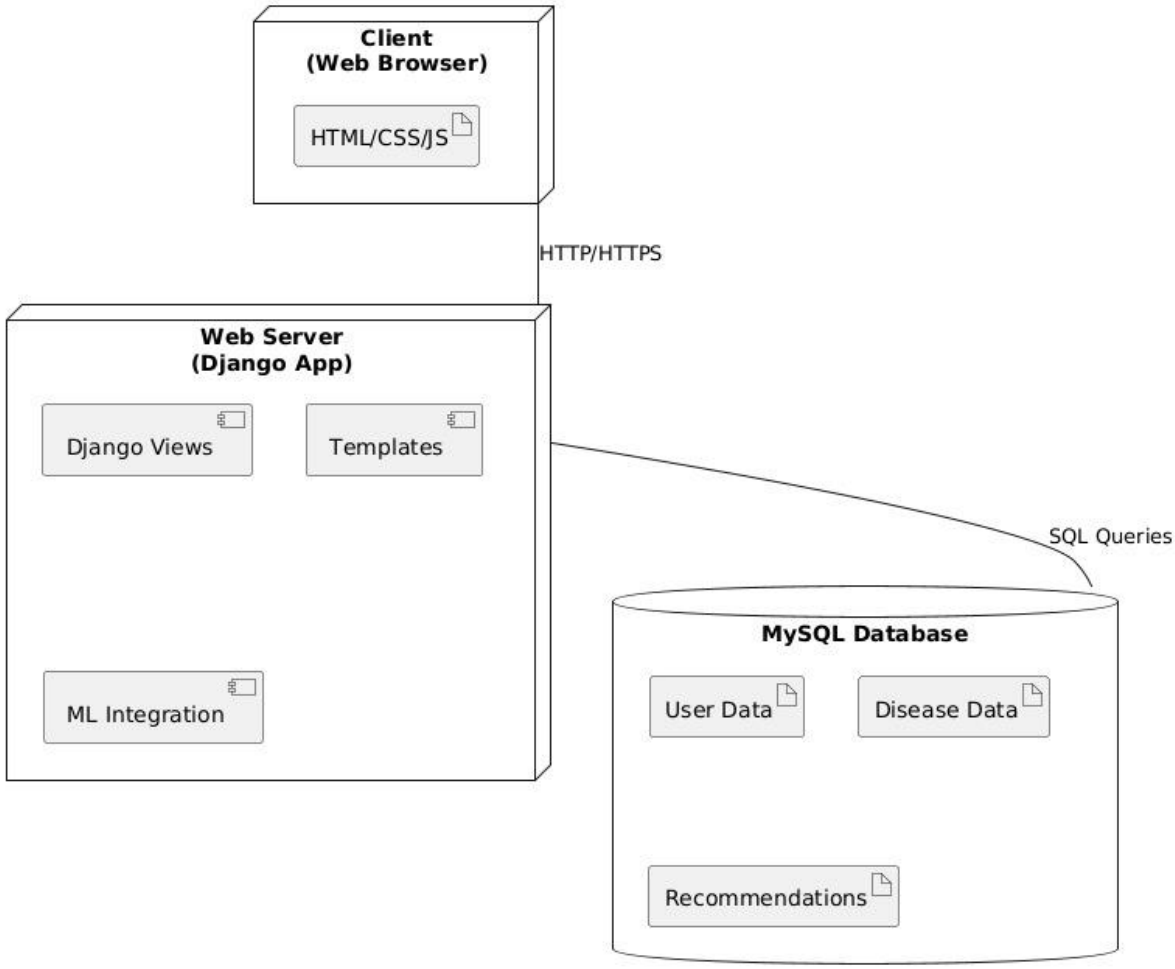
- Displays user-friendly messages if:
  - Database connection fails.
  - No symptoms are provided.
  - No matching disease found.

7. Assumptions

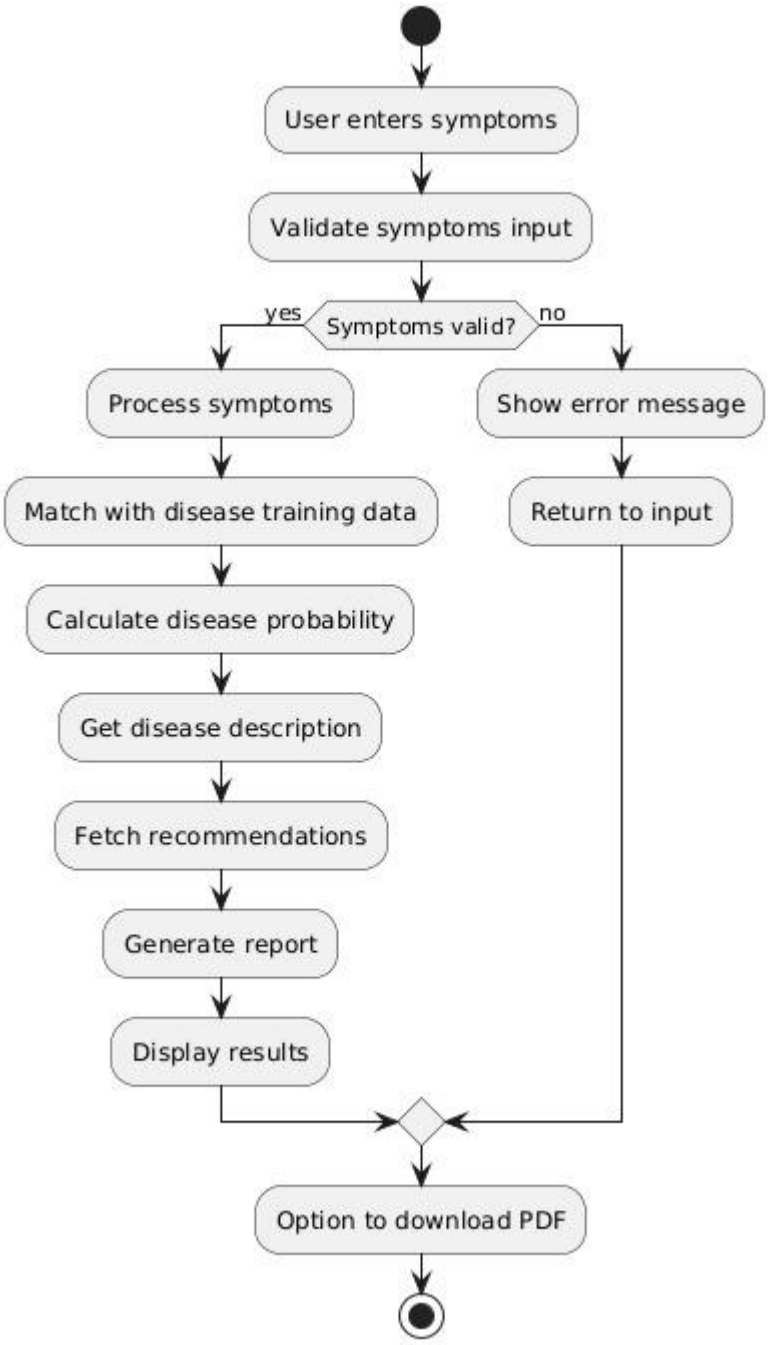
- The user knows how to type or speak symptoms properly.
- The database tables (users, symptom_severity, disease_training_data, disease_description, disease_diet, disease_medicine, disease_precaution, disease_workout) are already created and populated.
- The browser supports Speech Recognition API (Chrome preferred).

Software Engineering Paradigm

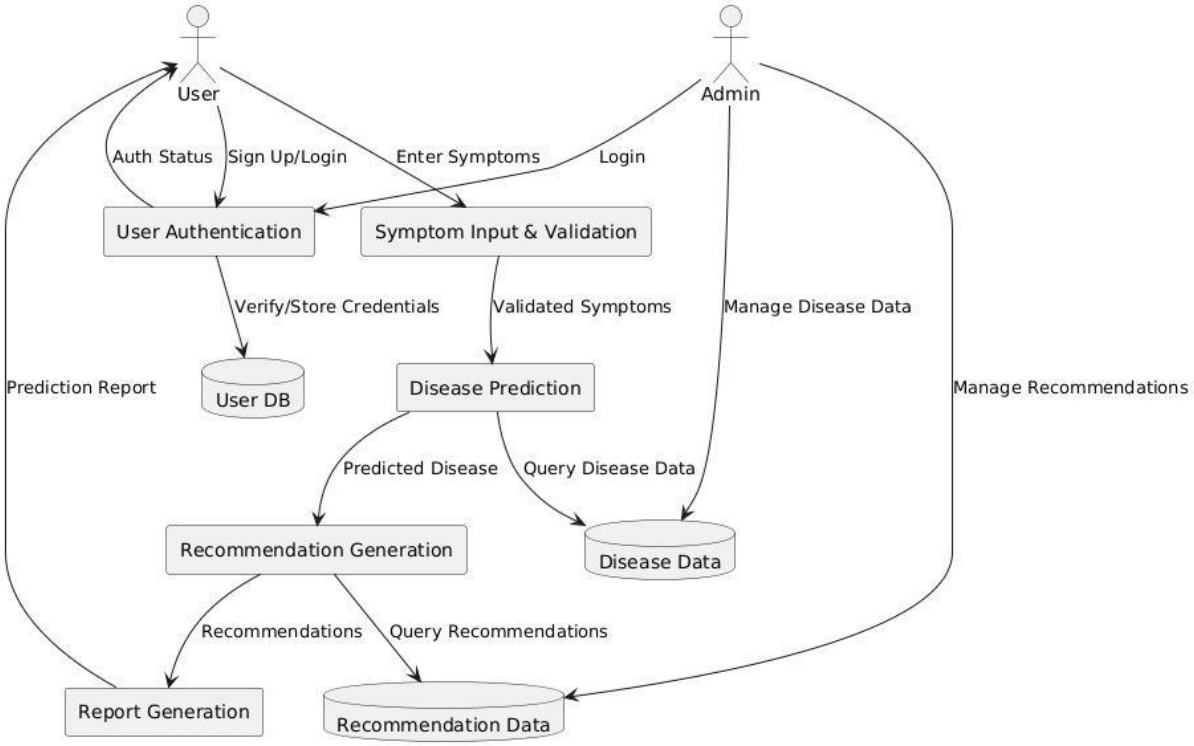
- Deployment Diagram



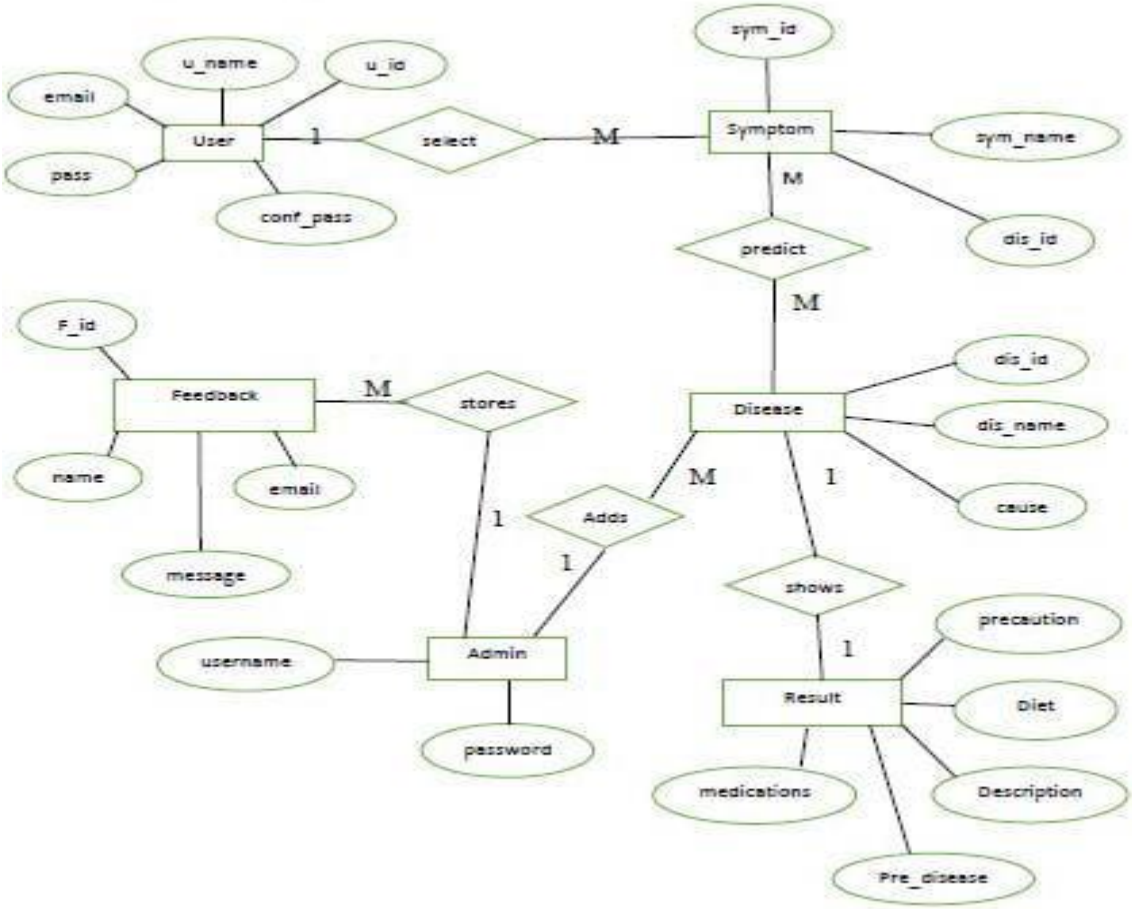
- **Activity Diagram for Disease Prediction Flow**



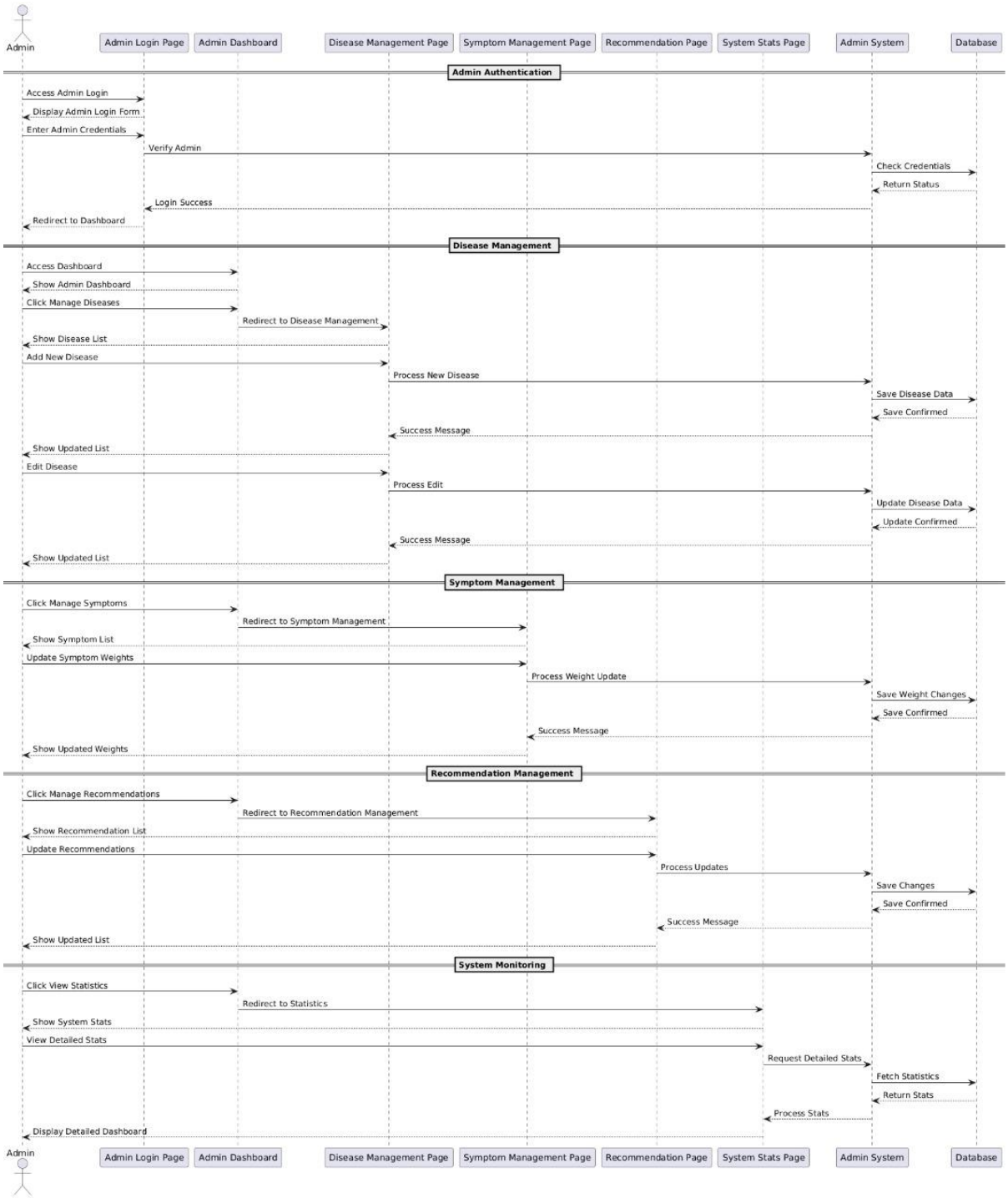
- **Data Flow Diagram of the Application**



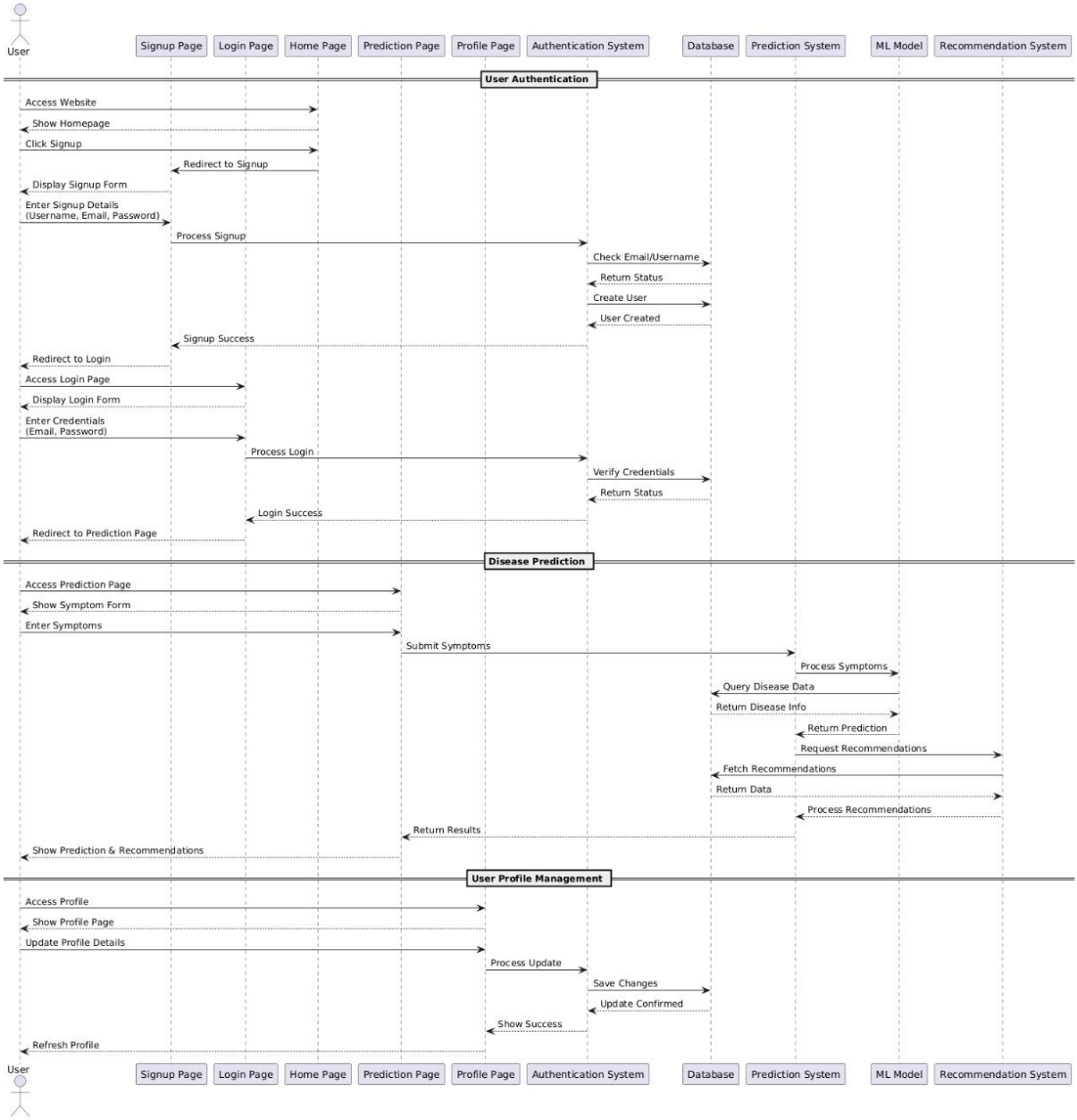
Entity-Relationship (ER) Diagram of the System



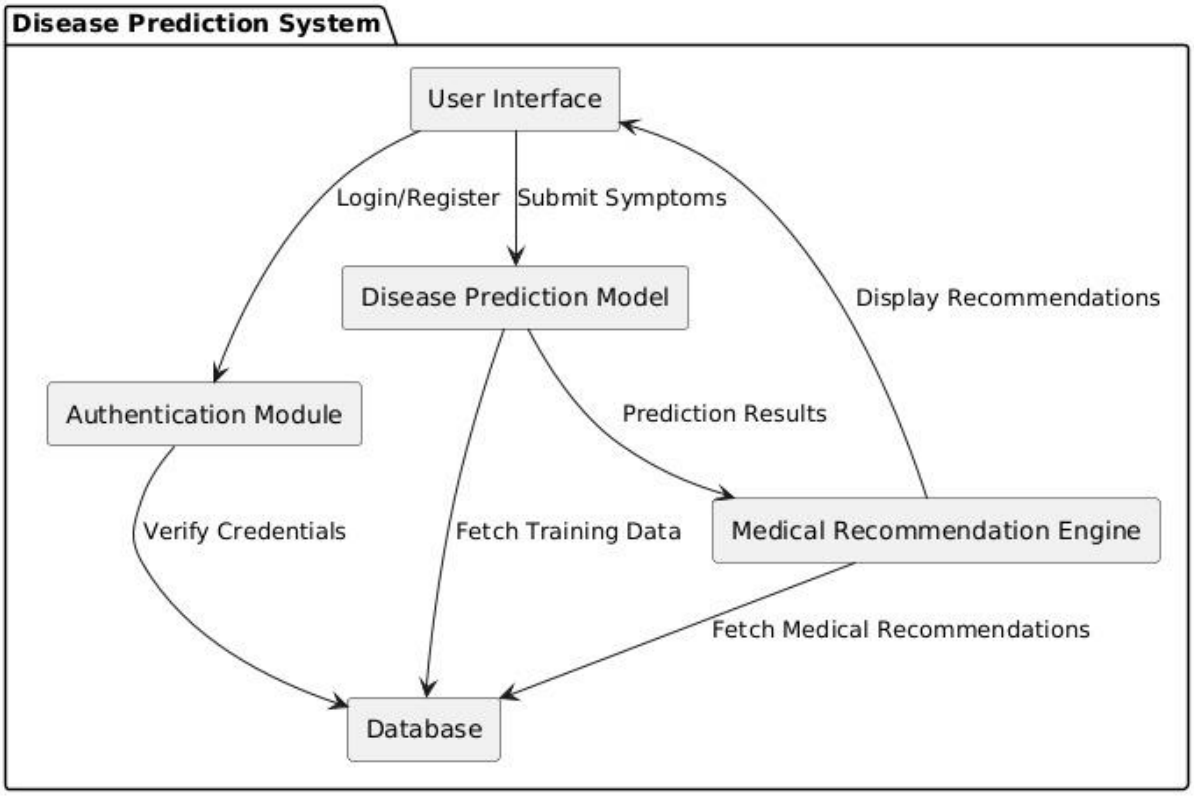
- **Sequence Diagram of System Communication Flow Admin**



User


User
Signup Page
Login Page
Home Page
Prediction Page
Profile Page
Authentication System
Database
Prediction System
ML Model
Recommendation System
User Authentication
Access Website
Show Homepage
Click Signup
Redirect to Signup
Display Signup Form
Enter Signup Details
(Username, Email, Password)
Process Signup
Check Email/Username
Return Status
Create User
User Created
Signup Success
Redirect to Login
Access Login Page
Display Login Form
Enter Credentials
(Email, Password)
Process Login
Verify Credentials
Return Status
Login Success
Redirect to Prediction Page
Disease Prediction
Access Prediction Page
Show Symptom Form
Enter Symptoms
Submit Symptoms
Process Symptoms
Query Disease Data
Return Disease Info
Return Prediction
Request Recommendations
Fetch Recommendations
Return Data
Process Recommendations
Return Results
Show Prediction & Recommendations
User Profile Management
Access Profile
Show Profile Page
Update Profile Details
Process Update
Save Changes
Update Confirmed
Show Success
Refresh Profile

- **Collaboration Diagram Representing System Modules Interaction**



Data Structure of all modules

1.Table disease_description_data

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
disease	varchar(100)	NO		NULL	
description	text	YES		NULL	

2. Table disease_diet_data

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
disease	varchar(100)	NO		NULL	
diet1	varchar(100)	YES		NULL	
diet2	varchar(100)	YES		NULL	
diet3	varchar(100)	YES		NULL	
diet4	varchar(100)	YES		NULL	

3. Table disease_medicine_data

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
disease	varchar(100)	NO		NULL	
medicine1	varchar(100)	YES		NULL	
medicine2	varchar(100)	YES		NULL	
medicine3	varchar(100)	YES		NULL	
medicine4	varchar(100)	YES		NULL	

4. Table disease_precaution_data

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
disease	varchar(100)	NO		NULL	
precaution1	varchar(100)	YES		NULL	
precaution2	varchar(100)	YES		NULL	
precaution3	varchar(100)	YES		NULL	
precaution4	varchar(100)	YES		NULL	

5. Table disease_training_data_data

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
symptom1	varchar(100)	YES		NULL	
symptom2	varchar(100)	YES		NULL	
symptom3	varchar(100)	YES		NULL	
symptom4	varchar(100)	YES		NULL	
symptom5	varchar(100)	YES		NULL	
disease	varchar(100)	YES		NULL	

6. Table symptom_severity_data

Field	Type	Null	Key	Default	Extra
id	bigint	NO	PRI	NULL	auto_increment
symptom	varchar(100)	NO	UNI	NULL	
weight	int	NO		NULL	

7. Table disease_workout_data

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
disease	varchar(100)	NO		NULL	
workout1	varchar(100)	YES		NULL	
workout2	varchar(100)	YES		NULL	
workout3	varchar(100)	YES		NULL	
workout4	varchar(100)	YES		NULL	

8. Table users_data

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
username	varchar(100)	NO		NULL	
email	varchar(100)	NO	UNI	NULL	
password	varchar(100)	NO		NULL	

Modules and Their Description for Disease Prediction and Medical Recommendation System

1. Django Application (manage.py)

- **Purpose**:
  Acts as the backend server to handle user requests (login, signup, prediction) and control the workflow of the system.
- **Key Functions**:
  - **home()**: Redirects logged-in users to the prediction page; otherwise to login.
  - **signup()**: Registers a new user in the database after checking if the username or email already exists.
  - **login()**: Authenticates user credentials and starts a session.
  - **logout()**: Ends the user session and redirects to login.
  - **predict()**: Handles disease prediction based on the entered symptoms, fetches disease details, and recommendations.

2. Database Connection (settings.py)

- **Purpose**:
  Provides a function (get_mysql_connection()) to connect the Django app to the MySQL database where all disease, symptoms, and user data are stored.
- **Key Usage**:
  - Connects to symptom_severity, disease_training_data, disease_description, disease_diet, disease_medicine, disease_precaution, and disease_workout tables.

3. Disease Prediction Logic (views.py → predict_disease)

- **Purpose**:
  Predicts the most likely disease based on entered symptoms by querying the training dataset and matches using MySQL queries.
- **Key Details**:
  - Fetches disease name based on symptom matching.
  - Fetches description, medications, diet, precautions, and workout recommendations associated with the predicted disease.

4. Frontend User Interface (index.html)

- **Purpose**:
  Provides a visually attractive web page for users to interact with the system.
- **Key Features**:
  - **Navbar**: Displays application title and user login status.
  - **Symptom Input Form**: Lets the user type or speak symptoms using speech recognition.
  - **Prediction Report**: Displays predicted disease details in a tabular format.
  - **PDF Download**: Allows users to download the prediction report as a PDF using the html2pdf.js library.
  - **Responsive Design**: Uses Bootstrap for mobile-friendly layout and design.

5. Speech Recognition Integration (index.html)

- **Purpose**:
  Allows users to input symptoms by speaking instead of typing.

- **Key Details**:
  - Uses the browser's built-in Speech Recognition API (Web Speech API) to convert voice input into text and fill it into the symptoms field.

6. Session Management (session.py)

- **Purpose**:
  Manages user authentication sessions (login/logout) to protect the prediction system from unauthorized access.
- **Key Details**:
  - Uses Django's session functionality to store the username after successful login.
  - Restricts access to /predict route unless logged in.

7. Error Handling

- **Purpose**:
  Ensures that minor issues like empty symptoms, database connection failures, or invalid user inputs do not crash the application.
- **Key Details**:
  - warnings.filterwarnings("ignore") used to suppress minor warnings.
  - Try-except blocks around database operations to catch and display errors gracefully.

Data Tables Used in the Project (Database)

- **users**: Stores registered users' username, email, and password.
- **symptom_severity**: List of symptoms.
- **disease_training_data**: Mapping of diseases to symptoms (training dataset).
- **disease_description**: Description of each disease.
- **disease_diet**: Recommended diets for each disease.
- **disease_medicine**: Suggested medicines for each disease.
- **disease_precaution**: Preventive measures for each disease.
- **disease_workout**: Recommended exercises/workouts for each disease.

Data structure of all modules :

1. Users Table (users)

- **username** (VARCHAR)
- **email** (VARCHAR)
- **password** (VARCHAR)

2. Symptom Severity Table (symptom_severity)

- **symptom** (VARCHAR)

3. Disease Training Data Table (disease_training_data)

- **disease** (VARCHAR)
- **symptom1** (VARCHAR)
- **symptom2** (VARCHAR)
- **symptom3** (VARCHAR)
- **symptom4** (VARCHAR)
- **symptom5** (VARCHAR)

4. Disease Description Table (disease_description)

- **disease** (VARCHAR)
- **description** (TEXT)

5. Disease Diet Table (disease_diet)

- **disease** (VARCHAR)
- **diet1** (VARCHAR)
- **diet2** (VARCHAR)
- **diet3** (VARCHAR)
- **diet4** (VARCHAR)

6. Disease Medicine Table (disease_medicine)

- **disease** (VARCHAR)
- **medicine1** (VARCHAR)
- **medicine2** (VARCHAR)
- **medicine3** (VARCHAR)
- **medicine4** (VARCHAR)

7. Disease Precaution Table (disease_precaution)

- **disease** (VARCHAR)
- **precaution1** (VARCHAR)
- **precaution2** (VARCHAR)
- **precaution3** (VARCHAR)
- **precaution4** (VARCHAR)

8. Disease Workout Table (disease_workout)

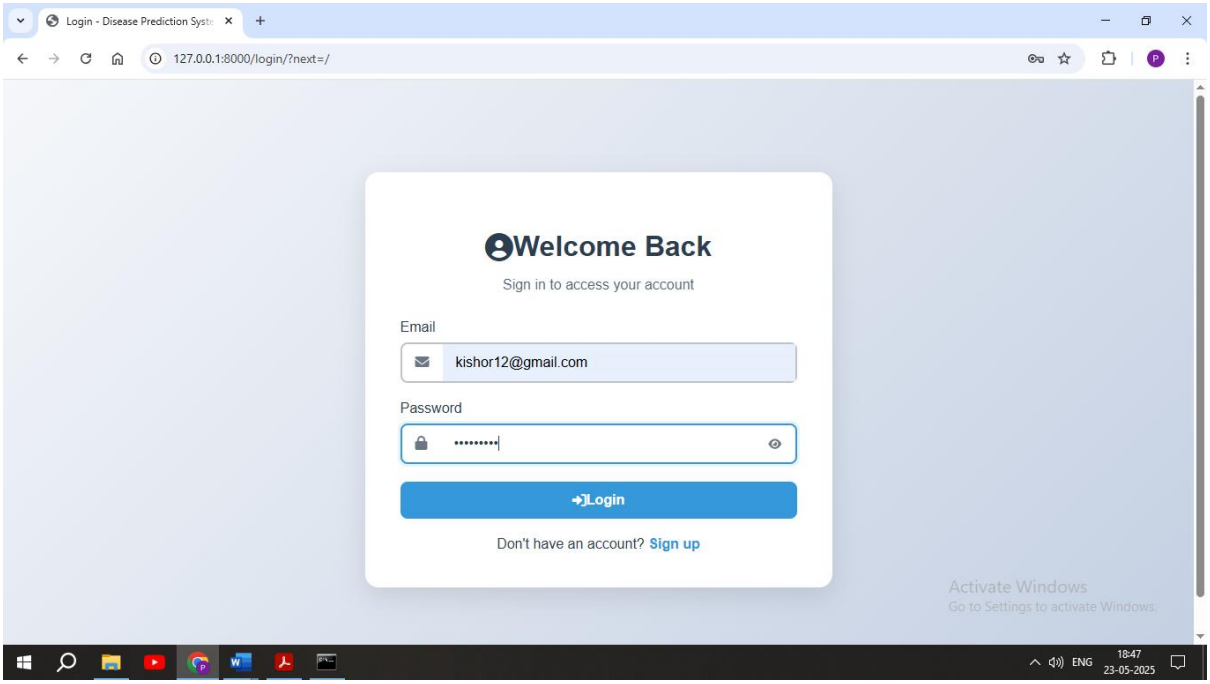
- **disease** (VARCHAR)
- **workout1** (VARCHAR)
- **workout2** (VARCHAR)
- **workout3** (VARCHAR)
- **workout4** (VARCHAR)

PROCEDURAL DESIGN:

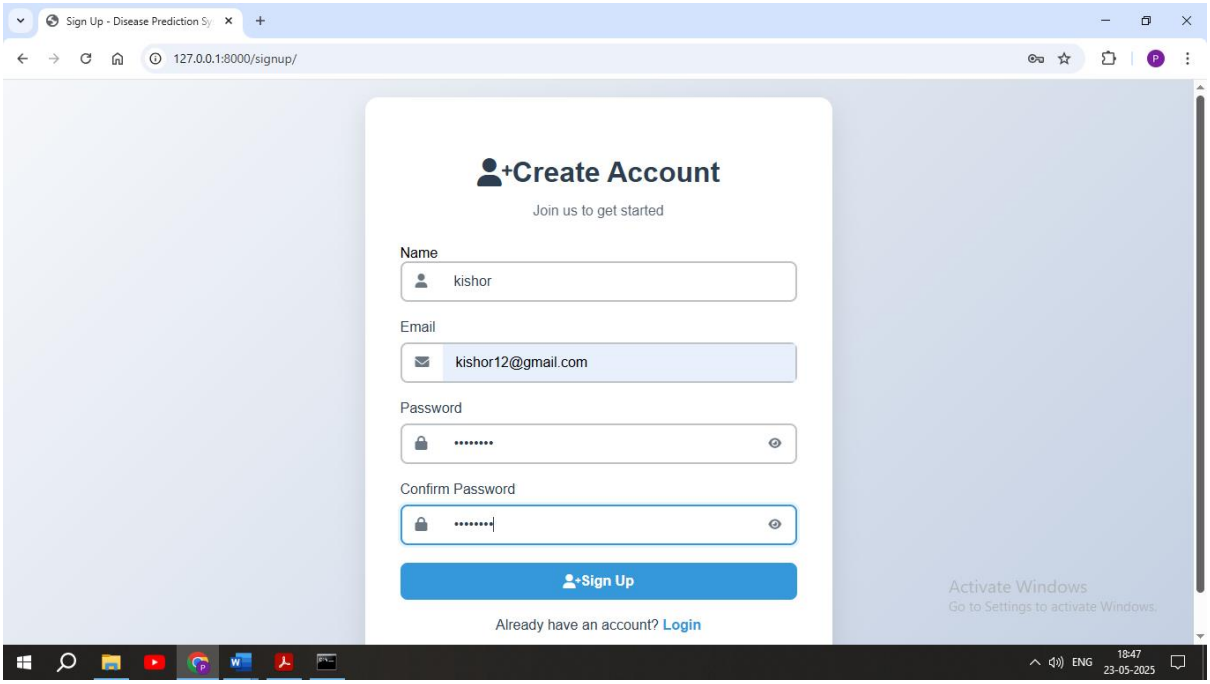
Table Name	Purpose
users	Store user registration data
symptom_severity	Store list of symptoms
disease_training_data	Store symptoms mapped to diseases
disease_description	Store description of diseases
disease_precaution	Store precautions for diseases
disease_medicine	Store medicines for diseases
disease_diet	Store diet plans for diseases
disease_workout	Store workout plans for diseases

Input Screens

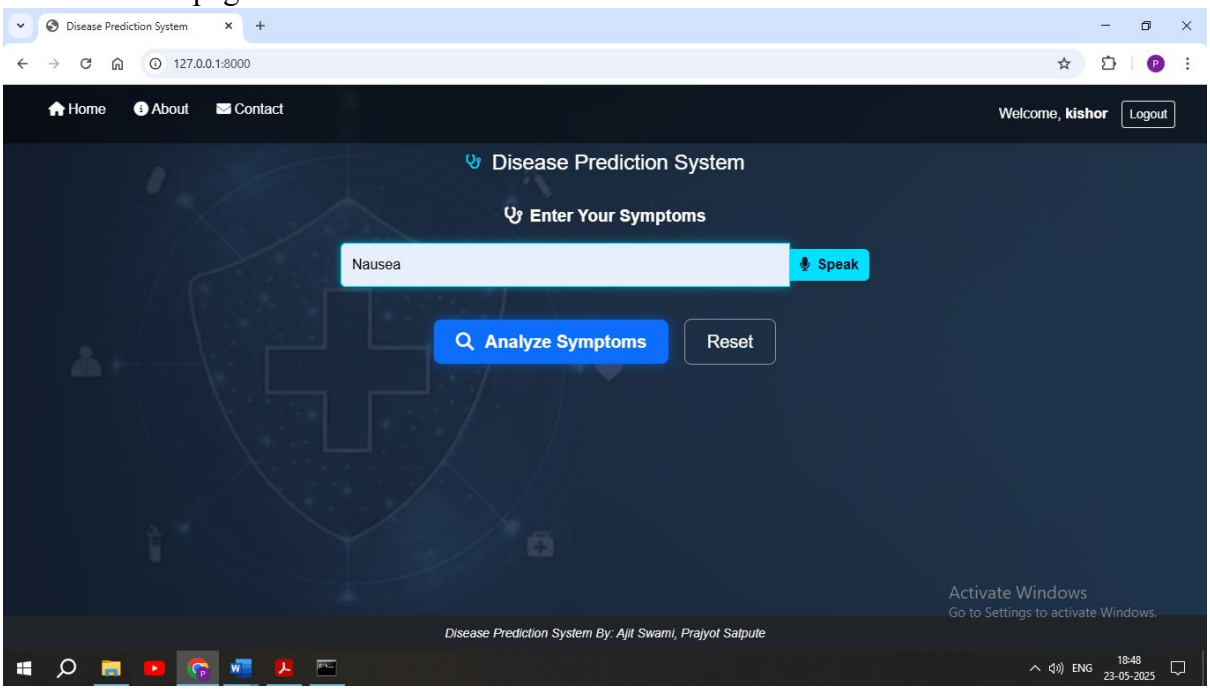
1. SIGNUP PAGE



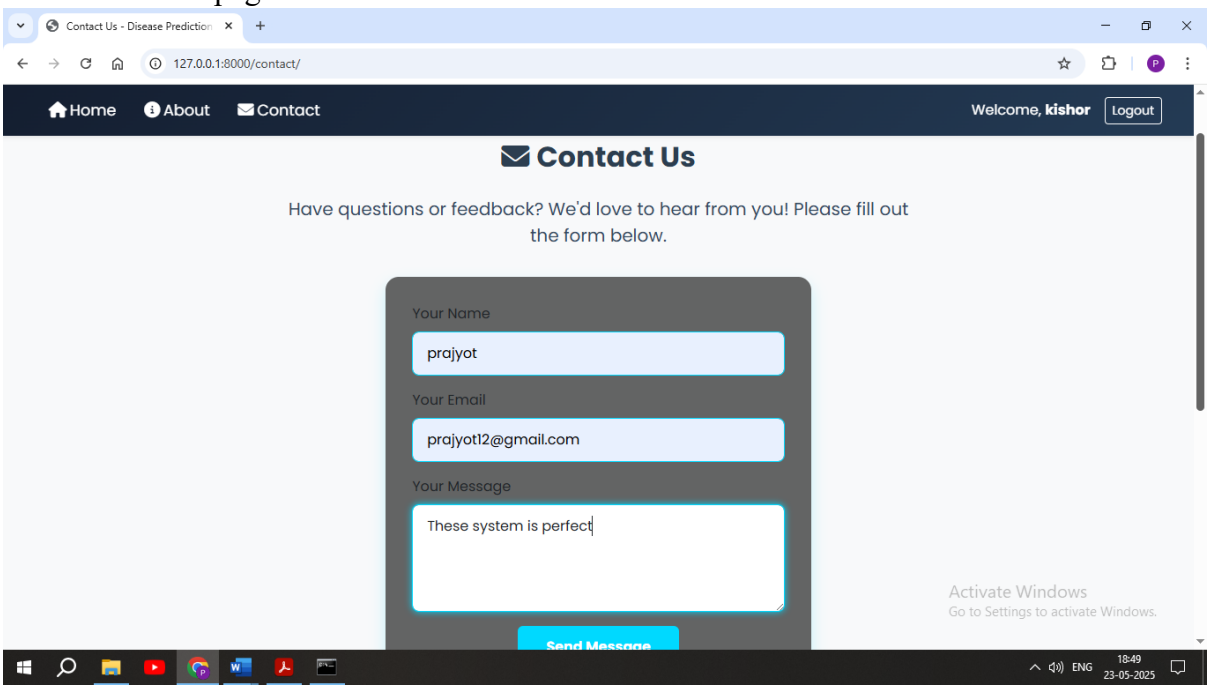
2. LOGIN PAGE



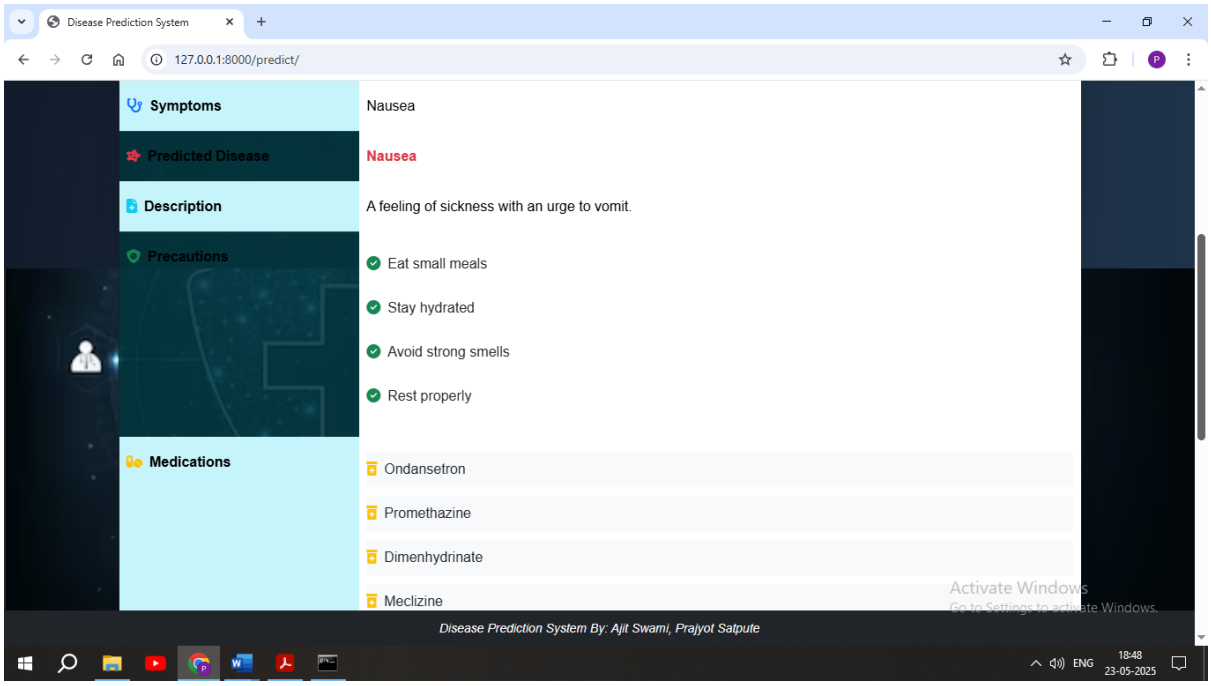
3. Home page

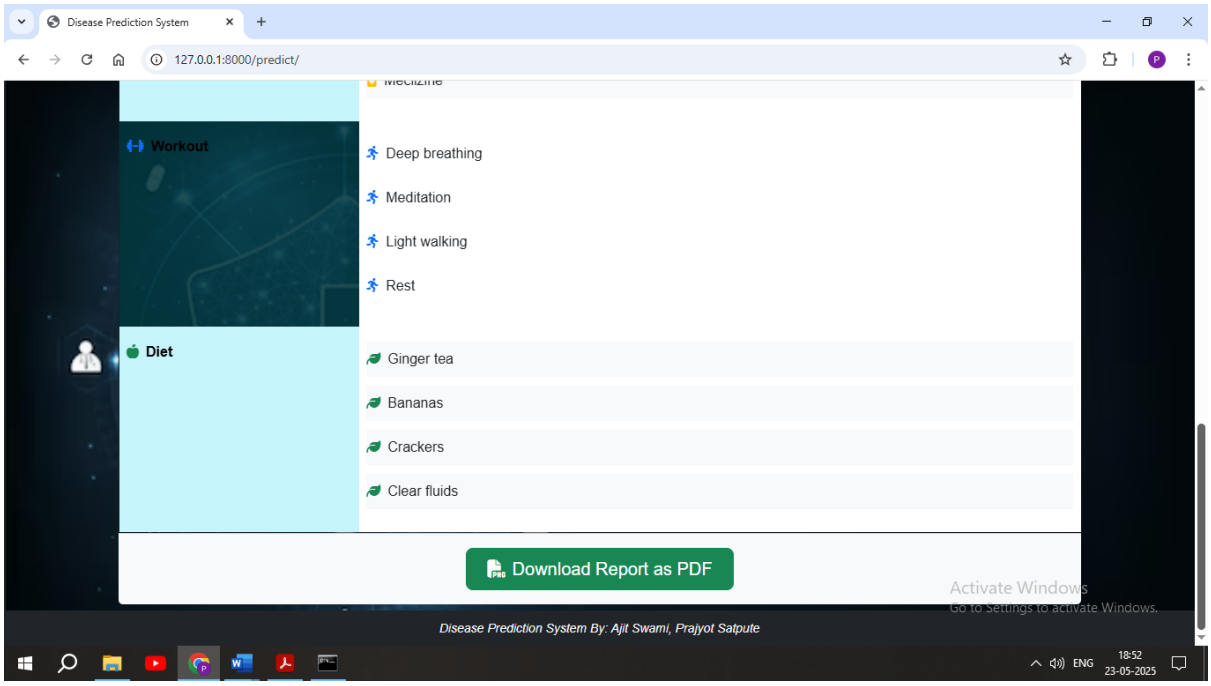


4. Contact page



Outputs of report:





CODING

SAMLE CODE:
Signup.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sign Up - Disease Prediction System</title>
    <link rel="stylesheet" href="{% static 'css/login.css' %}">
                    <link          rel="stylesheet"          href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.0.0/css/all.min.css">
</head>
<body>
    <div class="login-container">
        <div class="login-header">
            <h2><i class="fas fa-user-plus me-2"></i>Create Account</h2>
            <p>Join us to get started</p>
        </div>

        {% if messages %}
        <div class="messages">
            {% for message in messages %}
            <div class="alert alert-{{ message.tags }}">
                {{ message }}
            </div>
            {% endfor %}
        </div>
        {% endif %}

        <form method="POST" action="{% url 'signup' %}">
            {% csrf_token %}
            <div class="form-group">
                <label for="username">  Name</label>
                <div class="input-group">
                    <span class="input-group-text"><i class="fas fa-user"></i></span>
                            <input  type="text"  class="form-control"  id="username"  name="username"
placeholder="Enter your name" required>
                </div>
            </div>
```

```
        <div class="form-group">
          <label for="email" class="form-label">Email</label>
          <div class="input-group">
            <span class="input-group-text"><i class="fas fa-envelope"></i></span>
                    <input type="email" class="form-control" id="email" name="email"
placeholder="Enter your email" required>
          </div>
        </div>

        <div class="form-group">
          <label for="password1" class="form-label">Password</label>
          <div class="input-group">
            <span class="input-group-text"><i class="fas fa-lock"></i></span>
              <input type="password" class="form-control" id="password1" name="password1"
placeholder="Create a password" required>
                                  <button type="button" class="btn password-toggle"
onclick="togglePassword('password1')">
                <i class="fas fa-eye"></i>
            </button>
          </div>
        </div>

        <div class="form-group">
          <label for="password2" class="form-label">Confirm Password</label>
          <div class="input-group">
            <span class="input-group-text"><i class="fas fa-lock"></i></span>
              <input type="password" class="form-control" id="password2" name="password2"
placeholder="Confirm your password" required>
                                  <button type="button" class="btn password-toggle"
onclick="togglePassword('password2')">
                <i class="fas fa-eye"></i>
            </button>
          </div>
        </div>

        <div class="form-group">
          <button type="submit" class="btn btn-login">
            <i class="fas fa-user-plus me-2"></i>Sign Up
          </button>
        </div>

        <div class="signup-link">
          Already have an account? <a href="{% url 'login' %}">Login</a>
        </div>
      </form>
    </div>

    <script>
```

```
        function togglePassword(inputId) {
            const passwordInput = document.getElementById(inputId);
            const toggleButton = passwordInput.nextElementSibling.querySelector('i');

            if (passwordInput.type === 'password') {
                passwordInput.type = 'text';
                toggleButton.classList.remove('fa-eye');
                toggleButton.classList.add('fa-eye-slash');
            } else {
                passwordInput.type = 'password';
                toggleButton.classList.remove('fa-eye-slash');
                toggleButton.classList.add('fa-eye');
            }
        }
    </script>
</body>
</html>
```

Login.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login - Disease Prediction System</title>
    <link rel="stylesheet" href="{% static 'css/login.css' %}">
                    <link          rel="stylesheet"          href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.0.0/css/all.min.css">
</head>
<body>
    <div class="login-container">
        <div class="login-header">
            <h2><i class="fas fa-user-circle me-2"></i>Welcome Back</h2>
            <p>Sign in to access your account</p>
        </div>

        {% if messages %}
        <div class="messages">
            {% for message in messages %}
            <div class="alert alert-{{ message.tags }}">
                {{ message }}
            </div>
            {% endfor %}
        </div>
        {% endif %}
```

```
        <form method="POST" action="{% url 'login' %}">
            {% csrf_token %}
            <div class="form-group">
                <label for="email" class="form-label">Email</label>
                <div class="input-group">
                    <span class="input-group-text"><i class="fas fa-envelope"></i></span>
                                <input type="email" class="form-control" id="email" name="email"
placeholder="Enter your email" required>
                </div>
            </div>

            <div class="form-group">
                <label for="password" class="form-label">Password</label>
                <div class="input-group">
                    <span class="input-group-text"><i class="fas fa-lock"></i></span>
                        <input type="password" class="form-control" id="password" name="password"
placeholder="Enter your password" required>
                    <button type="button" class="btn password-toggle" onclick="togglePassword()">
                        <i class="fas fa-eye"></i>
                    </button>
                </div>
            </div>

            <div class="form-group">
                <button type="submit" class="btn btn-login">
                    <i class="fas fa-sign-in-alt me-2"></i>Login
                </button>
            </div>

            <div class="signup-link">
                Don't have an account? <a href="{% url 'signup' %}">Sign up</a>
            </div>
        </form>
    </div>

    <script>
        function togglePassword() {
            const passwordInput = document.getElementById('password');
            const toggleButton = document.querySelector('.password-toggle i');

            if (passwordInput.type === 'password') {
                passwordInput.type = 'text';
                toggleButton.classList.remove('fa-eye');
                toggleButton.classList.add('fa-eye-slash');
            } else {
                passwordInput.type = 'password';
                toggleButton.classList.remove('fa-eye-slash');
                toggleButton.classList.add('fa-eye');
```

```
        }
      }
    </script>
</body>
</html>
```

Index.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Disease Prediction System</title>

  <!-- Bootstrap CSS -->
  <link
   rel="stylesheet"
   href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
  />
  <!-- FontAwesome -->
  <link
   rel="stylesheet"
   href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.0/css/all.min.css"
  />
  <!-- AOS CSS -->
  <link rel="stylesheet" href="https://unpkg.com/aos@2.3.1/dist/aos.css" />
  <!-- Custom CSS -->
  <link rel="stylesheet" href="{% static 'css/index.css' %}" />
 </head>
 <body>
  <!-- Navbar -->
  <nav
   class="navbar navbar-dark bg-dark shadow-sm px-4 py-2"
   data-aos="fade-down"
  >
   <div
    class="container-fluid d-flex justify-content-between align-items-center"
   >
    <ul class="nav">
     <li class="nav-item">
      <a class="nav-link text-white" href="{% url 'predict' %}">
       <i class="fas fa-home me-1"></i>Home
      </a>
```

```
      </li>
      <li class="nav-item">
        <a class="nav-link text-white" href="{% url 'about' %}">
          <i class="fas fa-info-circle me-1"></i>About
        </a>
      </li>
      <li class="nav-item">
        <a class="nav-link text-white" href="{% url 'contact' %}">
          <i class="fas fa-envelope me-1"></i>Contact
        </a>
      </li>
    </ul>

    <div class="d-flex align-items-center">
      {% if user.is_authenticated %}
      <span class="navbar-text text-white me-3">
        Welcome, <strong>{{ user.username }}</strong>
      </span>
      <a class="btn btn-sm btn-outline-light" href="{% url 'logout' %}"
        >Logout</a
      >
      {% else %}
      <a class="btn btn-sm btn-outline-light" href="{% url 'login' %}"
        >Login</a
      >
      {% endif %}
    </div>
  </div>
</nav>

<!-- Main Content -->
<div class="container">
  <!-- Main Heading Section with Typing Effect -->
  <div class="container text-center py-2" data-aos="fade-up">
    <a class="navbar-brand mx-auto" href="{% url 'predict' %}">
      <i class="fas fa-stethoscope text-info me-2"></i>
      <span class="h4 align-middle typing-text"
        >Disease Prediction System</span
      >
    </a>
  </div>

  <!-- Form Section where user enters symptoms -->
  <div class="container my-4" data-aos="fade-up" data-aos-delay="100">
    <form id="predictForm" action="{% url 'predict' %}" method="POST">
      {% csrf_token %}
      <div class="form-group mb-3">
        <div class="d-flex justify-content-center">
```

```
    <label for="symptoms" class="form-label fw-bold text-center">
      <i class="fas fa-stethoscope me-2"></i>Enter Your Symptoms
    </label>
  </div>

  <div class="input-group justify-content-center align-items-center">
    <input
      type="text"
      class="form-control rounded-start"
      id="symptoms"
      name="symptoms"
      placeholder="e.g. headache, anxiety"
      required
    />
    <button
      type="button"
      class="btn btn-info rounded-end"
      onclick="startListening()"
    >
      <i class="fas fa-microphone me-2"></i>Speak
    </button>
  </div>
</div>

<div class="text-center mt-3">
  <button
    type="submit"
    class="btn btn-primary btn-lg me-2"
    id="submitBtn"
  >
    <i class="fas fa-search me-2"></i>
    <span
      class="spinner-border spinner-border-sm d-none"
      role="status"
      aria-hidden="true"
      id="loadingSpinner"
    ></span>
    <span id="loadingText" class="d-none">Processing...</span>
    <span id="predictText">Analyze Symptoms</span>
  </button>
  <button
    type="button"
    class="btn btn-outline-secondary btn-lg"
    onclick="resetForm()"
  >
    Reset
  </button>
</div>
```

```
  </form>
</div>

<!-- Results Section -->
{% if predicted_disease %}
<div class="container mb-5" id="results-container" data-aos="zoom-in-up">
  <div class="card border-0 shadow-lg">
    <div class="card-header bg-dark text-white p-3">
      <h3 class="text-center mb-0">
        <i class="fas fa-clipboard-list me-2"></i>Medical Analysis Report
      </h3>
    </div>

    <div class="card-body p-0">
      <div class="table-responsive">
        <table class="table table-hover mb-0 results-table">
          <tbody>
            <tr class="bg-light">
              <th class="py-3" style="width: 25%">
                <i class="fas fa-stethoscope me-2 text-primary"></i
                >Symptoms
              </th>
              <td class="py-3">{{ symptoms }}</td>
            </tr>
            <tr>
              <th class="py-3">
                <i class="fas fa-disease me-2 text-danger"></i>Predicted
                Disease
              </th>
              <td class="py-3 fw-bold text-danger">
                {{ predicted_disease }}
              </td>
            </tr>
            <tr class="bg-light">
              <th class="py-3">
                <i class="fas fa-file-medical me-2 text-info"></i
                >Description
              </th>
              <td class="py-3">
                {{ dis_des|default:"No description available" }}
              </td>
            </tr>
            <tr>
              <th class="py-3">
                <i class="fas fa-shield-virus me-2 text-success"></i
                >Precautions
              </th>
              <td class="py-3">
```

```
    <ul class="list-group list-group-flush">
      {% for i in my_precautions %}
      <li class="list-group-item border-0 ps-0">
        <i class="fas fa-check-circle text-success me-2"></i
        >{{ i }}
      </li>
      {% endfor %}
    </ul>
  </td>
</tr>
<tr class="bg-light">
  <th class="py-3">
    <i class="fas fa-pills me-2 text-warning"></i>Medications
  </th>
  <td class="py-3">
    <ul class="list-group list-group-flush">
      {% for i in medications %}
      <li class="list-group-item border-0 bg-light ps-0">
        <i
          class="fas fa-prescription-bottle-alt text-warning me-2"
        ></i
        >{{ i }}
      </li>
      {% endfor %}
    </ul>
  </td>
</tr>
<tr>
  <th class="py-3">
    <i class="fas fa-dumbbell me-2 text-primary"></i>Workout
  </th>
  <td class="py-3">
    <ul class="list-group list-group-flush">
      {% for i in workout %}
      <li class="list-group-item border-0 ps-0">
        <i class="fas fa-running text-primary me-2"></i>{{ i }}
      </li>
      {% endfor %}
    </ul>
  </td>
</tr>
<tr class="bg-light">
  <th class="py-3">
    <i class="fas fa-apple-alt me-2 text-success"></i>Diet
  </th>
  <td class="py-3">
    <ul class="list-group list-group-flush">
      {% for i in my_diet %}
```

```
                    <li class="list-group-item border-0 bg-light ps-0">
                      <i class="fas fa-leaf text-success me-2"></i>{{ i }}
                    </li>
                    {% endfor %}
                  </ul>
                </td>
              </tr>
            </tbody>
          </table>
        </div>
      </div>

      <div class="card-footer text-center p-3 bg-light">
        <button onclick="downloadPDF()" class="btn btn-success btn-lg px-4">
          <i class="fas fa-file-pdf me-2"></i>Download Report as PDF
        </button>
      </div>
    </div>
  </div>
  {% endif %}
</div>

<!-- Footer -->
<footer class="footer bg-dark text-light py-2">
  <div class="container">
    <div class="text-center">
      <small
        ><i
          >Disease Prediction System By: Ajit Swami, Prajyot Satpute</i
        ></small
      >
    </div>
  </div>
</footer>

<!-- Scripts -->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/html2pdf.js/0.9.3/html2pdf.bundle.min.js"></script>
<script src="https://unpkg.com/aos@2.3.1/dist/aos.js"></script>

<script>
  // Initialize AOS
  AOS.init();

  // Function to generate and download the PDF report
  function downloadPDF() {
```

```
        const element = document.getElementById("results-container");
        html2pdf(element, {
          margin: 10,
          filename: `Disease_Prediction_Report_${new Date()
            .toISOString()
            .slice(0, 10)}.pdf`,
          image: { type: "jpeg", quality: 0.98 },
          html2canvas: { scale: 2 },
          jsPDF: { unit: "mm", format: "a4", orientation: "portrait" },
        });
      }

      // Start Speech Recognition to fill the symptoms input field
      function startListening() {
        if (
          !("webkitSpeechRecognition" in window) &&
          !("SpeechRecognition" in window)
        ) {
          alert("Speech recognition is not supported in this browser.");
          return;
        }
        const recognition = new (window.SpeechRecognition ||
          window.webkitSpeechRecognition)();
        recognition.lang = "en-US";
        recognition.onresult = (event) => {
          const transcript = event.results[0][0].transcript;
          document.getElementById("symptoms").value = transcript;
        };
        recognition.start();
      }

      // Reset the form
      function resetForm() {
        document.getElementById("predictForm").reset();
      }
    </script>
  </body>
</html>
```

Views.py

```
"""
Disease Prediction System - Views
This file contains all the view functions for handling HTTP requests and responses.
Includes authentication, prediction, and recommendation views.
"""

from django.shortcuts import render, redirect
```

```
from django.contrib.auth import login, authenticate, logout
from django.contrib.auth.decorators import login_required
from django.contrib.auth.models import User
from django.db.models import Count
from django.contrib import messages
from .models import (
    SymptomSeverity, DiseaseTraining, DiseaseDescription,
    DiseaseDiet, DiseaseMedicine, DiseasePrecaution, DiseaseWorkout,

)
from fuzzywuzzy import process
from django.views.decorators.csrf import csrf_protect
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier

def home(request):
    """
    Home page view.
    Renders the main landing page of the application.
    """
    return render(request, 'home.html')

def load_symptoms():
    """Load available symptoms from the database."""
    symptoms = SymptomSeverity.objects.values_list('symptom', flat=True)
    return {symptom.lower(): symptom for symptom in symptoms}

def predict_disease(patient_symptoms):
    """Predict disease based on input symptoms."""
    if not patient_symptoms:
        return "No symptoms provided", {}, {}

    # Query to find diseases with matching symptoms
    diseases = DiseaseTraining.objects.filter(
        symptom1__in=patient_symptoms
    ).values('disease').annotate(
        symptom_matches=Count('disease')
    ).order_by('-symptom_matches')[:1]

    if not diseases:
        return "Unknown Disease", {}, {}

    disease = diseases[0]['disease']
    disease_data = {}
    recommendations = {}
```

```
    # Get disease description
    try:
        description = DiseaseDescription.objects.get(disease=disease)
        disease_data['description'] = description.description
    except DiseaseDescription.DoesNotExist:
        disease_data['description'] = "No description available."

    # Get diet recommendations
    try:
        diet = DiseaseDiet.objects.get(disease=disease)
        recommendations['diet'] = [
            item for item in [diet.diet1, diet.diet2, diet.diet3, diet.diet4]
            if item and item.strip()
        ]
    except DiseaseDiet.DoesNotExist:
        recommendations['diet'] = ["No diet recommendations."]

    # Get medicine recommendations
    try:
        medicine = DiseaseMedicine.objects.get(disease=disease)
        recommendations['medicine'] = [
                item for item in [medicine.medicine1, medicine.medicine2, medicine.medicine3,
medicine.medicine4]
            if item and item.strip()
        ]
    except DiseaseMedicine.DoesNotExist:
        recommendations['medicine'] = ["No medicine recommendations."]

    # Get precautions
    try:
        precaution = DiseasePrecaution.objects.get(disease=disease)
        recommendations['precautions'] = [
             item for item in [precaution.precaution1, precaution.precaution2, precaution.precaution3,
precaution.precaution4]
            if item and item.strip()
        ]
    except DiseasePrecaution.DoesNotExist:
        recommendations['precautions'] = ["No precautions available."]

    # Get workout recommendations
    try:
        workout = DiseaseWorkout.objects.get(disease=disease)
        recommendations['workout'] = [
                    item for item in [workout.workout1, workout.workout2, workout.workout3,
workout.workout4]
            if item and item.strip()
        ]
    except DiseaseWorkout.DoesNotExist:
```

```
        recommendations['workout'] = ["No workout recommendations."]

    return disease, disease_data, recommendations

def signup(request):
    """
    User registration view.
    Handles new user registration and account creation.
    """
    if request.method == 'POST':
        username = request.POST.get('username')
        email = request.POST.get('email')
        password1 = request.POST.get('password1')
        password2 = request.POST.get('password2')

        # Validate input
        if not username or not email or not password1 or not password2:
            messages.error(request, "All fields are required.")
            return render(request, 'signup.html')

        if password1 != password2:
            messages.error(request, "Passwords do not match.")
            return render(request, 'signup.html')

        if len(password1) < 8:
            messages.error(request, "Password must be at least 8 characters long.")
            return render(request, 'signup.html')

        # Check if username or email already exists
        if User.objects.filter(username=username).exists():
            messages.error(request, "Username already exists.")
            return render(request, 'signup.html')

        if User.objects.filter(email=email).exists():
            messages.error(request, "Email already exists.")
            return render(request, 'signup.html')

        try:
            # Create new user
            user = User.objects.create_user(
                username=username,
                email=email,
                password=password1
            )
            messages.success(request, "Account created successfully! Please login.")
            return redirect('login')
        except Exception as e:
            messages.error(request, f"Error creating account: {str(e)}")
```

```
            return render(request, 'signup.html')

    return render(request, 'signup.html')

def login_view(request):
    """
    User login view.
    Handles user login and authentication.
    """
    if request.method == 'POST':
        email = request.POST.get('email')
        password = request.POST.get('password')

        try:
            # First try to get user by email
            user = User.objects.get(email=email)
            # Then authenticate with username and password
            authenticated_user = authenticate(request, username=user.username, password=password)

            if authenticated_user is not None:
                login(request, authenticated_user)
                return redirect('predict')
            else:
                messages.error(request, "Invalid password. Please try again.")
        except User.DoesNotExist:
            messages.error(request, "No account found with this email. Please sign up first.")
        except Exception as e:
            messages.error(request, f"An error occurred: {str(e)}")

    return render(request, 'login.html')

def logout_view(request):
    logout(request)
    return redirect('login')

@login_required
@csrf_protect
def predict_view(request):
    """
    Disease prediction view.
    Handles symptom input and disease prediction using ML model.
    Returns prediction results and recommendations.
    """
    if request.method == 'POST':
        symptoms = request.POST.get('symptoms', '').split(',')
        symptoms = [s.strip() for s in symptoms if s.strip()]

        predicted_disease, disease_data, recommendations = predict_disease(symptoms)
```

```
            return render(request, 'index.html', {
                'symptoms': ', '.join(symptoms),
                'predicted_disease': predicted_disease,
                'dis_des': disease_data.get('description', 'No description.'),
                'my_diet': recommendations.get('diet', []),
                'my_precautions': recommendations.get('precautions', []),
                'medications': recommendations.get('medicine', []),
                'workout': recommendations.get('workout', []),
                'prediction_source': "Symptom Analysis"
            })
        return render(request, 'index.html')

    def about(request):
        """
        About page view.
        Displays information about the application and its features.
        """
        return render(request, 'about.html')

    def contact(request):
        """
        Contact page view.
        Provides contact information and support details.
        """
        return render(request, 'contact.html')

    def view_users(request):
        """View to display all registered users."""
        if not request.user.is_superuser:
            messages.error(request, "Only administrators can view user list.")
            return redirect('home')

        users = User.objects.all().order_by('date_joined')
        return render(request, 'users.html', {'users': users})


    settings.py
    """
    Django settings for healthcare_project project.
    """

    from pathlib import Path
    import os

    # Build paths inside the project like this: BASE_DIR / 'subdir'.
    BASE_DIR = Path(__file__).resolve().parent.parent
```

```
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = '79d0fd91612b21b0b1148576816f9b87'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'disease_prediction',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'healthcare_project.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'healthcare_project.wsgi.application'
```

```
# Database
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'disease_data',
        'USER': 'root',
        'PASSWORD': 'Prajyot@53',  # Change this to your actual MySQL password
        'HOST': 'localhost',
        'PORT': '3306',
    }
}

# Password validation
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Internationalization
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_TZ = True

# Static files (CSS, JavaScript, Images)
STATIC_URL = 'static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]

# Default primary key field type
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

# Login URL
LOGIN_URL = '/login/'
LOGIN_REDIRECT_URL = '/predict/'
```

Error Handling :

1. **Database Connection Failure**
    - If get_mysql_connection() returns None, display a "Database connection failed" message.
    - Avoid running further queries if the database is not connected.
2. **Symptom Input Validation**
    - Check if user has entered symptoms.
    - If no symptoms are entered, show "Please enter at least one symptom."
3. **Unknown Symptoms**
    - If entered symptoms do not match database symptoms, return "Unknown symptoms" or ask user to re-enter properly.
4. **Disease Not Found**
    - If no disease matches the entered symptoms, show "No matching disease found."
5. **Database Query Failures**
    - Catch exceptions during SQL query execution.
    - If any error occurs (e.g., wrong query, missing tables), return a friendly error message like "Data fetch error. Please try again later."
6. **Empty Data Handling**
    - If description, medicines, diets, precautions, or workouts are missing for a predicted disease, show "No data available" instead of leaving fields empty.
7. **Login Errors**
    - If wrong email or password entered, show "Invalid credentials. Please try again."
    - If database error during login, inform "Login service temporarily unavailable."
8. **Signup Errors**
    - If username or email already exists, show "Username or Email already registered."
    - If any database insertion error occurs, show "Signup failed. Please try again."
9. **Session Expiry / Unauthorized Access**
    - If user tries to access /predict without logging in, redirect to /login page with message "Please login first."
10. **Speech Recognition Failure**
    - If browser does not support Speech Recognition, show alert "Speech recognition not supported in this browser."
11. **PDF Download Error**
    - If prediction results are missing and user tries to download PDF, disable the PDF button or alert "No report available to download."
12. **Unexpected Server Errors**
    - Use try-except blocks wherever possible to catch unexpected errors and show a general "Something went wrong, please try again later."

Code Improvement

1. Security and Best Practices

- **Password Security**: Store hashed passwords instead of plain text.
- **SQL Injection Protection**: You're already using %s, but double-check inputs.
- **Session Security**: Set session cookie attributes (e.g., SESSION_COOKIE_SECURE, SESSION_COOKIE_HTTPONLY).

2. Code Structure and Efficiency

- **Create a models.py**: Move all DB functions (predict_disease, load_symptoms, etc.) into a separate file.
- **Create a routes.py**: Keep all routes organized separately.
- **Error Handling**: More graceful error pages (404, 500) instead of basic print statements.

3. Front-End (index.html)

- **Loading Indicator**: Show a spinner when predicting disease (better UX).
- **Better Voice Recognition Error Handling**: Handle microphone permission denials.
- **Make it Responsive**: It's Bootstrap, so it's mostly good, but a little more padding/margin tweaks.
- **Auto-clear Symptoms Box** after submitting or successful prediction.

4. Performance

- **Connection Pooling**: Instead of opening and closing MySQL connection every time, use connection pooling (e.g., mysql.connector.pooling).
- **Reduce Queries**: Optimize SQL by using joins if possible to fetch all disease-related data in one query.

5. Feature Upgrades (Optional but impactful!)

- **Add Multiple Disease Predictions**: Right now, it predicts only the top match. You could show top 3 matching diseases.
- **Allow file uploads (optional)**: Upload previous reports to assist prediction.
- **Auto-Suggestions on Symptoms**: As user types, auto-suggest symptoms (frontend improvement).
- **Admin Panel**: To manage diseases, medicines, precautions, etc.
- **Dark/Light Theme Toggle**: Fancy but users love this.

Area	Improvements
Security	Hash passwords, manage sessions safely
Database	Connection pooling, optimize SQL queries
Backend	Modularize (models/routes separation)
Frontend	Loading spinner, error handling, better UX
Features	Multiple disease suggestions, admin dashboard, upload reports, etc.
Performance	Reduce DB hits, optimize HTML generation

PARAMETER PASSING

1. User → Frontend (index.html)

- **User types symptoms** in the <input> box.
- When user clicks "Predict", the **form sends data** using **POST method** to /predict route.

Form Data:
- symptoms = (e.g., "headache, fever")

2. Frontend → Backend (Django @app.route('/predict', methods=['POST']))

- Django **receives** the symptoms entered by the user.
- It **reads** them with:

symptoms = request.form.get('symptoms')

- It **splits** the symptoms into a list:

symptoms = [s.strip() for s in symptoms.split(',')]

Example:
Input: "headache, fever"
After splitting: ['headache', 'fever']

3. Backend → Database (MySQL)

- Django **calls** the predict_disease(symptoms) function.
- The function **queries** the database using the list of symptoms.
- It **finds**:
    - The **predicted disease**,
    - **Description** of the disease,
    - **Precautions**, **Medications**, **Workouts**, **Diet**.

Example result:
Disease: Dengue
Description: "Dengue is a mosquito-borne viral disease."
Precautions: ["Use mosquito net", "Wear long sleeves"]
Medications: ["Paracetamol", "Drink fluids"]
Workout: ["Light stretching"]
Diet: ["Papaya leaf juice", "Hydration"]

4. Backend → Frontend (Passing Data to Template)

- Django **sends** the prediction results to index.html:

```
return render_template('index.html',
    symptoms=', '.join(symptoms),
    predicted_disease=predicted_disease,
    dis_des=disease_data.get('description', 'No description.'),
    my_diet=recommendations.get('diet', []),
    my_precautions=recommendations.get('precautions', []),
    medications=recommendations.get('medicine', []),
    workout=recommendations.get('workout', [])
)
```

These variables are passed:

- symptoms
- predicted_disease
- dis_des (description)
- my_diet (diet list)
- my_precautions (precaution list)
- medications (medication list)
- workout (workout list)

5. Frontend (index.html) → Show Results

- index.html **receives** the variables and **displays** them inside a table:

Example:

<td>{{ symptoms }}</td> → Shows user-entered symptoms
<td>{{ predicted_disease }}</td> → Shows predicted disease
<td>{{ dis_des }}</td> → Shows description
{% for i in my_precautions %}
{{ i }} → Shows list of precautions
{% endfor %}

Summary Chart:

Step	Who	Action
1	User	Enters symptoms and submits form
2	Browser	Sends symptoms to Django backend (/predict)
3	Django	Receives symptoms, queries database
4	Django	Sends back prediction + details
5	Browser	Displays disease, description, medicines, diet, etc.

VALIDATION CHECKS

Form Input Validation (Frontend in index.html and Backend in main.py)

- Check if **symptoms input** is not empty before submitting.
- Allow only **letters, commas, and spaces** in symptoms field (no numbers/special characters).
- Limit **symptoms field length** (e.g., max 250 characters) to avoid overflow.
- Trim leading/trailing spaces in symptoms before sending to server.
- Prevent **SQL Injection** by using parameterized queries (✅ already done).

Signup Form Validation (Backend in signup route)

- Ensure **username**, **email**, and **password** fields are not empty.
- Validate **email format** (e.g., using regex like something@example.com).
- Ensure **password is strong** (e.g., minimum 6 characters).
- Check if **username/email already exists** before inserting (✅ already done).
- Sanitize inputs to avoid unwanted characters.

Login Form Validation (Backend in login route)

- Ensure **email** and **password** fields are not empty.
- Validate **email format** before querying database.
- Show clear error message if login fails (✅ already handled).

Database Validation

- Ensure **no NULL values** are inserted in users table fields.
- Enforce **unique constraints** on **username** and **email** fields in database.
- Validate **symptoms** entered exist in your symptom_severity table (optional but good).

Session Validation

- Protect /predict page - only **logged in users** can access (✅ already done).
- Prevent unauthorized access to prediction or logout routes.

Additional Suggestions

- Add **error handling** if database connection fails.
- Flash friendly **error/success messages** on signup/login pages.
- **Limit file download attempts** if no prediction was made yet.

TESTING

Testing Strategy for Disease Prediction System

1. Unit Testing

- **Test Database Connection**
  - Check if MySQL database connects successfully.
  - Handle database connection failures gracefully.
- **Test load_symptoms() Function**
  - Ensure it loads all symptoms correctly from symptom_severity table.
  - Handle empty or missing data cases.
- **Test predict_disease(patient_symptoms) Function**
  - Test prediction with valid symptoms.
  - Test prediction with unknown symptoms.
  - Test when no symptoms are provided.
  - Verify correct retrieval of description, precautions, medications, diet, and workout.
- **Test User Authentication**
  - Test user signup with new credentials.
  - Test signup with already existing username or email.
  - Test login with correct credentials.
  - Test login with incorrect credentials.
  - Test logout functionality.

2. Integration Testing

- **Test /signup, /login, /logout, /predict Routes**
  - Ensure the user flows (Signup → Login → Predict → Logout) work smoothly.
  - Check session management: user cannot access /predict without logging in.
- **Form Submission**
  - Submit symptoms through the form and verify correct disease prediction.
  - Submit empty form and check for proper error handling.
- **PDF Download**
  - After prediction, test the "Download Report as PDF" button.
  - Ensure PDF is generated with correct information.
- **Speech-to-Text Input**
  - Test the microphone button for symptom input via speech recognition.

3. UI Testing

- **Responsive Design**
  - Test that all pages look good on desktop, tablet, and mobile devices.
- **Navbar and Footer**
  - Check if Navbar shows the logged-in username and logout button properly.
  - Footer should always display project credits.
- **Error Messages**
  - Signup/login errors should be clear (e.g., "Username already exists" or "Invalid credentials").

4. Security Testing

- **Session Handling**
  - Ensure sessions expire properly after logout.
- **Input Validation**
  - Validate form inputs to avoid SQL injection and bad data.
- **Password Protection**
  - (Improvement Tip) Currently, passwords are stored in plain text — use hashing (like bcrypt) in production.

5. Performance Testing

- **Load Testing**
  - Test with multiple users predicting diseases at the same time.
- **Database Query Time**
  - Measure and optimize the time taken for fetching disease data.

6. Manual Testing Checklist

Test Case	Expected Result
Signup with new user	Account created
Signup with existing email	Error message
Login with correct credentials	Redirect to predict page
Login with wrong credentials	Show error
Predict disease with valid symptoms	Show prediction report
Predict disease with unknown symptoms	Show "Unknown Disease"
Generate PDF report	Download PDF correctly
Speak symptoms via mic	Fill input field automatically
Logout	Redirect to login page

SYSTEM SECURITY

1. Password Security

- **Hashing Passwords**: Store passwords securely by hashing them before saving to the database. Use libraries like werkzeug.security to hash and verify passwords instead of storing plain text passwords.
- from werkzeug.security import generate_password_hash, check_password_hash
-
- # For signup
- hashed_password = generate_password_hash(password)
-
- # For login
- if check_password_hash(user[2], password): # Assuming the password is stored at index 2
- session['username'] = user[1]

2. Session Management

- **Session Timeout**: Set an expiration time for user sessions to avoid long-lived sessions. You can use the PERMANENT_SESSION_LIFETIME attribute to control session expiration.
- from datetime import timedelta
- app.permanent_session_lifetime = timedelta(minutes=30)
- **Secure Cookies**: Use secure cookies for storing session data by enabling the SESSION_COOKIE_SECURE flag in production (to prevent session hijacking).
- app.config['SESSION_COOKIE_SECURE'] = True # Use HTTPS in production

3. SQL Injection Prevention

- **Parameterized Queries**: Use parameterized queries to avoid SQL injection attacks. You're already doing this with cursor.execute(), but ensure that you're always using it with placeholders (e.g., %s) to avoid direct string formatting.
- cursor.execute("SELECT * FROM users WHERE email = %s AND password = %s", (email, password))

4. Cross-Site Scripting (XSS) Prevention

- **Sanitizing Inputs**: Ensure that user inputs are sanitized and validated before being displayed on the webpage. You can use libraries like bleach for HTML sanitization, or simply escape special characters in the inputs.
- Django's render_template automatically escapes variables by default. However, when displaying data directly, ensure you're escaping potentially harmful content using Markup from Django:
- from Django import Markup
- safe_content = Markup(user_input)

5. Cross-Site Request Forgery (CSRF) Protection

- **CSRF Tokens**: Django uses Django-WTF to help protect against CSRF attacks. By using Django-WTF, you automatically get protection against CSRF by adding a CSRF token to all forms.
- from Django_wtf.csrf import CSRFProtect
- csrf = CSRFProtect(app)
- For forms in your templates, ensure you include the {{ csrf_token() }} inside each form.

- <form method="POST">
- {{ csrf_token() }}
- <!-- form fields -->
- </form>

6. Secure HTTP Headers

- **Strict-Transport-Security (HSTS)**: Ensure that your web application is only accessed over HTTPS by setting the HTTP header for Strict-Transport-Security.
- @app.after_request
- def apply_security_headers(response):
- response.headers['Strict-Transport-Security'] = 'max-age=31536000; includeSubDomains'
- return response

7. File Upload Security

- If you plan to allow file uploads (e.g., PDFs), ensure the uploaded files are validated for type and size to prevent malicious files from being uploaded. You can also store files outside the web root to avoid direct access.
- ALLOWED_EXTENSIONS = {'pdf', 'docx', 'png', 'jpg'}
-
- def allowed_file(filename):
- return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

8. Database Security

- **Use Database Roles and Permissions**: Ensure that the database user has restricted privileges (read-only where necessary) to minimize the impact of a compromised application.
- **Backup Data Regularly**: Ensure your database is backed up regularly to prevent data loss in case of an attack or failure.

9. Logging and Monitoring

- **Log Security Events**: Implement logging for sensitive operations (like login attempts) and monitor them for suspicious activity.
- Use logging libraries and consider using services like Sentry or CloudWatch for real-time monitoring.

10. Rate Limiting

To avoid brute-force attacks, you can use Django-Limiter to limit the number of requests per user (IP or session).

- from Django_limiter import Limiter
- limiter = Limiter(app)
-
- @limiter.limit("5 per minute")
- @app.route('/login', methods=['POST'])
- def login():
- # login code

11. Application Dependency Updates

- **Update Libraries**: Regularly check for security updates to the libraries you are using (like Django, MySQL connectors, etc.). You can use tools like pip-audit or Safety to identify vulnerabilities in your dependencies.

Prediction Report

Symptoms	headache, fever
Predicted Disease	Flu
Description	The flu is a contagious respiratory illness caused by influenza viruses that infect the nose, throat, and sometimes the lungs
Precautions	• Get annual flu vaccination • Avold close contact with sick individuals • Wash hands frequently • Cover mouth and nose when coughing or sneezing
Medications	• Oseltamivir • Zanamivir
Workout	• Rest and recover • Gentle stretching • Deep breathing exercises
Diet	• Stay hydrated with clear fluids • Eat nutrient-rich foods • Avoid alcohol and caffeine

FUTURE SCOPE

While the current implementation of the **Disease Prediction and Medical Recommendation System** serves as a reliable prototype for primary health guidance, there is significant potential to improve and expand its capabilities. This section outlines various enhancements and research directions that could further increase the system's **usefulness, scalability, accuracy**, and **accessibility** in real-world scenarios.

1. Expansion of Knowledge Base

Currently, the system supports a limited set of diseases and symptoms. In the future, more comprehensive and diverse datasets can be integrated, covering:

- Chronic illnesses (e.g., diabetes, hypertension)
- Mental health issues (e.g., anxiety, depression)
- Rare diseases
- Gender-specific and age-specific conditions

This would allow the system to support a broader range of users with more personalized advice.

2. Multi-Language Support

To make the system accessible to non-English speakers, especially in rural or underserved regions, multi-language support can be introduced. Natural Language Processing (NLP) libraries can be used to detect and respond in various Indian and international languages.

3.Integration with Machine Learning Models

Future versions of the system can include supervised learning models (like Decision Trees, Random Forest, or Naive Bayes) trained on a larger dataset to provide more precise predictions. These models can evolve over time through continuous learning based on user interactions.

4.Mobile App Development

Creating an Android/iOS app version would make the system more portable and user-friendly. Offline capabilities (limited symptom matching) can also be introduced for areas with low internet connectivity.

5. Personalized Recommendations

With user consent, the system can be enhanced to provide personalized advice by collecting basic health profile details such as:

- Age
- Gender
- Medical history
- Allergies or past medications

This would improve accuracy and relevance of the medical recommendations.

6.Real-Time Doctor Consultation Integration

By integrating telemedicine APIs or chat-based consultation platforms, users can be offered the option to connect with licensed healthcare professionals if their symptoms are serious.
Enhanced Privacy and Security Measures

To comply with global healthcare data regulations such as **HIPAA** or **GDPR**, future versions can implement:

- Encrypted data storage
- Consent-based data logging
- Clear disclaimers and privacy policies

7.Collaboration with Medical Experts

Medical professionals can be involved in periodically reviewing and updating the system's recommendations, ensuring accuracy and clinical safety.

Future Enhancement

1. **User Authentication Improvements**
   - Implement social media login (Google, Facebook).
   - Add multi-factor authentication for better security.
2. **Machine Learning Integration**
   - Integrate a trained ML model to improve disease prediction accuracy.
   - Add functionality to periodically retrain the model with new data.
3. **Mobile Responsiveness**
   - Use responsive CSS frameworks like Bootstrap or Tailwind to enhance UI on mobile devices.
4. **User Dashboard**
   - Provide users with a dashboard showing their past reports, suggestions, and improvements in health metrics.
5. **Doctor/Admin Panel**
   - Develop a dedicated admin panel where doctors can view and analyze user data and provide feedback.
6. **API Development**
   - Build RESTful APIs for integration with mobile apps or third-party health services.
7. **Health Tips & Blogs Section**
   - Provide users with dynamic content like daily health tips, disease prevention blogs, etc.
8. **Data Visualization**
   - Use libraries like Chart.js or D3.js to display user health data trends and patterns.
9. **Appointment Booking System**
   - Integrate a module for booking appointments with doctors or consultants directly from the app.
10. **Notification System**
    - Add email or SMS notifications for test results, upcoming appointments, or health reminders.
11. **Internationalization (i18n)**
    - Support multiple languages for a wider reach, especially in multilingual countries.
12. **Security Enhancements**
    - Enforce HTTPS, CSRF protection, input validation, and secure database storage practices.
13. **Performance Optimization**
    - Use caching (e.g., Redis) to speed up the site, especially for repeated queries or heavy pages.
14. **User Feedback System**
    - Let users give feedback on prediction accuracy and application usability to drive improvements.
15. **Data Export**
    - Allow users to export their health data in formats like PDF or Excel for personal use or doctor consultations.

BIBLIOGRAPHY

1. **ResearchGate – Artificial Intelligence in Healthcare: A Review**
   https://www.researchgate.net/publication/380536893_Artificial_Intelligence_in_Healthcare_A_Review

2. **Journal of Pharmaceutical Research International – Disease Diagnosis Using ML** https://journaljpri.com/index.php/JPRI/article/view/2637

3. **ResearchGate – Chatbot for Healthcare System Using AI**
   https://www.researchgate.net/publication/344983576_Chatbot_for_Healthcare_System_Using_Artificial_Intelligence

4. **Journal of Medical Internet Research – Health Care Chatbots Analysis**
   https://www.jmir.org/2024/1/e56930/

5. **ResearchGate – Predicting Disease from Symptoms Using ML**
   https://www.researchgate.net/publication/374909594_Predicting_disease_from_several_symptoms_using_machine_learning_approach

6. **Official Flask Documentation** https://flask.palletsprojects.com

7. **MySQL Official Documentation** https://dev.mysql.com/doc/

8. **FuzzyWuzzy String Matching Library – GitHub**
   https://github.com/seatgeek/fuzzywuzzy

9. **WebMD – Medical Condition Reference** https://www.webmd.com

10. **Healthline – Symptom Checker & Health Guide**
    https://www.healthline.com