

# **Well Bot – Global Wellness Assistant Chatbot:**

## **Project Documentation**

### **Under the Guidance of**

Dr K. Santhiya

### **Team Members**

1. Meghana Macharla
2. Aishwarya polagoni
3. Akash Rawat
4. Gopichand Morla

# 1. Project Overview

## Problem Statement

Access to **reliable, non-diagnostic health and wellness information** is often fragmented, language-dependent, or behind paywalls, creating a significant barrier to proactive self-care, especially in underserved global communities. Existing online advice can be misleading or lack authoritative grounding. The core problem WellBot addresses is the need for a **secure, universally accessible, multilingual conversational tool** that delivers evidence-based, public health-aligned guidance (sourced from organizations like WHO and CDC) directly to users through an intuitive, non-medical interface.

### 1.1. Introduction and Purpose

This report details the design, development, and implementation of **WellBot**, a multilingual, AI-powered conversational agent designed to serve as a global wellness assistant. The primary objective of WellBot is to democratize access to reliable, non-diagnostic health and wellness information by leveraging modern conversational AI technologies. It provides personalized, self-help guidance inspired by authoritative sources, promoting proactive health management for its users.

### 1.2. Scope and Audience

The system's scope includes secure user management, a robust, multilingual conversational engine, and an intuitive web interface backed by a structured knowledge base.

- **Functional Scope:** Includes user authentication, personalized profile settings, real-time multilingual chat, knowledge retrieval, and administrative control over content and feedback.
- **Non-Functional Scope:** Focuses on security (JWT), scalability (Rasa/Flask microservices), responsiveness (UI/UX), and data persistence (SQLite).

The primary users are the **general public** seeking non-diagnostic wellness advice. Secondary audiences include the project **developers and maintainers**, and the **system administrators** managing the knowledge base.

### 1.3. Primary Technology Stack

WellBot is built on a modular, open-source technology stack, ensuring flexibility, maintainability, and clear separation of concerns between the web server, database, and conversational AI.

- **Backend & Framework (Flask - Python):** Acts as the RESTful API server, handling authentication and request routing.
- **Authentication (Flask-JWT-Extended):** Manages secure token-based authentication using HTTP-only cookies for persistent sessions.
- **Conversational AI (Rasa - NLU + Core):** Provides Natural Language Understanding (NLU) for intent/entity recognition and Dialogue Management (Core) for conversation flow.
- **Database (SQLite):** A local, file-based database used for persistent storage of user profiles and the central health knowledge base.
- **ORM (SQLAlchemy):** The Object-Relational Mapper facilitating efficient, abstract, and secure Python-based database interactions.
- **Frontend (HTML, CSS, JavaScript):** Handles client-side logic for user interfaces (login, registration, chat) and dynamic interaction with the Flask API.

### 1.4. Guiding Principles: Public Health Reference

The core content of WellBot's knowledge base is synthesized from guidelines provided by reputable public health organizations, including the **World Health Organization (WHO)** and the **Centers for Disease Control and Prevention (CDC)**. This foundational principle ensures that all guidance provided is non-diagnostic, evidence-based, and aligned with global public health standards.

## 2. System Architecture

WellBot employs a **Three-Layer Architectural Model** to ensure decoupling and scalability. The system is composed of four main components communicating via HTTP: the **Flask Web Server** (handles user authentication and routing), the **Rasa Server** (processes NLP and dialogue flow), the **Rasa Action Server** (executes custom business logic and database queries), and the **SQLite Database** (persistent data storage). Requests flow securely from the frontend, authenticated by Flask, passed to Rasa for processing, and finally routed to the Action Server to retrieve multilingual knowledge.

### 2.1. Three-Layer Architectural Model

WellBot operates on a clear, three-layer architecture to maximize decoupling, scalability, and maintainability:

- **Presentation Layer (Frontend):** Handles all user interactions (login, chat, profile updates).
- **Application Layer (Flask/Rasa Core):** Manages business logic, authentication, request routing, and dialogue flow.
- **Data Layer (SQLite/SQLAlchemy/Rasa Action Server):** Responsible for persistent data storage and secure knowledge retrieval.

### 2.2. Component Breakdown

Each component plays a specialized role and communicates via defined protocols:

- **User Interface:** Renders UI templates, captures user input, and initiates AJAX requests. It communicates using HTTP/S (to Flask).
- **Flask Web Server (Port 5000):** Responsible for user authentication, JWT issuance, session validation, static file serving, and proxying chat requests to Rasa. It communicates via HTTP/S (External) and HTTP (Internal to Rasa).

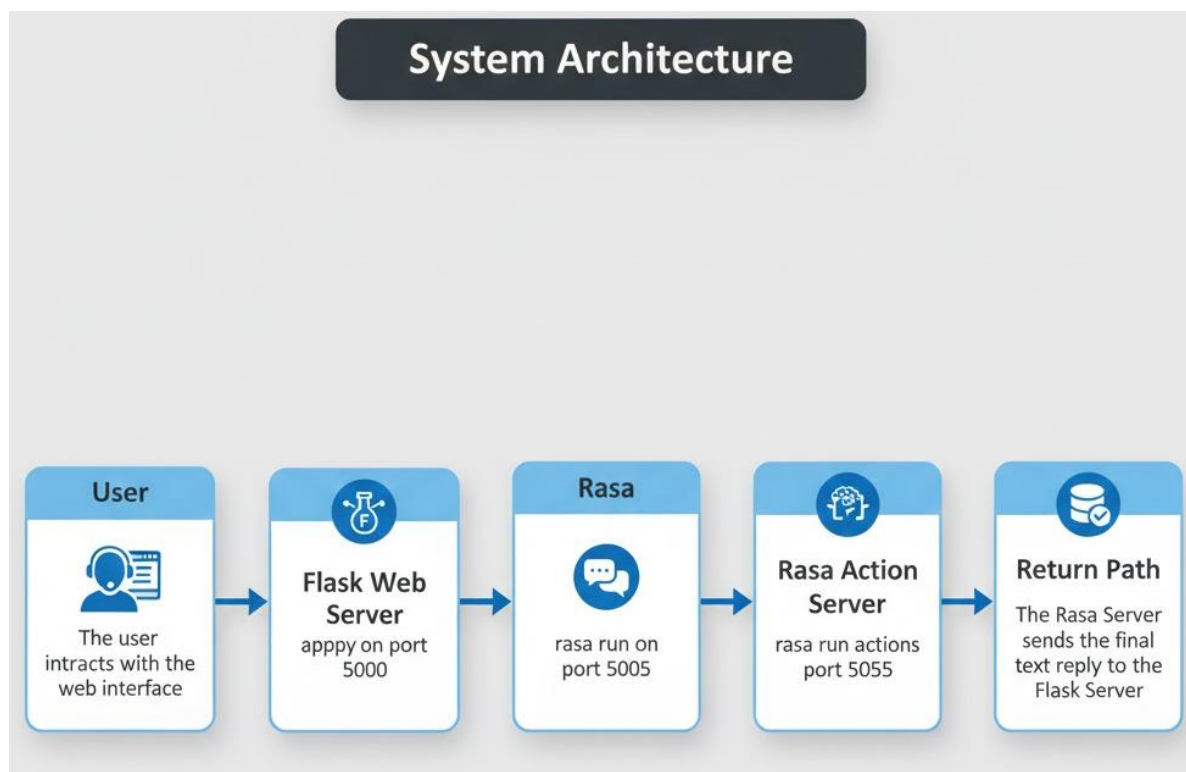
- **Rasa Server (Port 5005):** Processes raw text (NLU), manages dialogue state (Core), and determines the appropriate next action. It communicates via HTTP (Internal to Flask/Action Server).
- **Rasa Action Server (Port 5055):** Executes custom business logic, primarily the database query for the health knowledge base. It communicates via HTTP (Internal to Rasa) and SQLAlchemy ORM (to SQLite).
- **SQLite Database:** Provides persistent storage for User and HealthKnowledge data, accessed through the SQLAlchemy ORM.

### 2.3. Data and Request Flow Diagram (Conceptual)

The chat interaction workflow follows this critical sequential process:

1. **User Sends Message:** The Frontend captures user input and sends a POST request to Flask's /chat endpoint. The request includes the message and the secured JWT Cookie.
2. **Flask Authentication:** Flask validates the JWT. If valid, it extracts the user\_id and the stored preferred\_language.
3. **Flask to Rasa:** Flask forwards the user message and the language metadata (as a custom channel or slot update) to the Rasa server.
4. **Rasa NLU Processing:** Rasa NLU detects the conversation's intent (e.g., symptom\_inquiry) and extracts relevant entities (e.g., duration, symptom).
5. **Rasa Core Dialogue:** Rasa Core determines the next action. For knowledge queries, it triggers the custom action: action\_query\_knowledge\_base.
6. **Rasa to Action Server:** Rasa sends a request to the Action Server.
7. **Action Server Execution:** The action\_query\_knowledge\_base uses SQLAlchemy to query the SQLite database, utilizing the detected intent and the user's preferred language to retrieve the correct multilingual response.
8. **Response Return:** The structured response (in the correct language) travels back through the Rasa Server, then to the Flask Server.

9. **Flask to Frontend:** Flask relays the final response to the frontend, where JavaScript dynamically updates the chat.html interface.
- 



### 3. Milestone 1 – Core Foundation: Authentication & User Profiles

This milestone established the **secure foundation** of the application. It focuses on implementing **robust user management** by defining the database schema, creating secure user registration with **password hashing**, and integrating **token-based session management (JWT)** using HTTP-only cookies to protect user sessions and enforce access control across the web application.

#### 3.1. Database Schema: User Management

The persistence layer for user data is managed by SQLAlchemy and stored in SQLite. The primary table, **User**, is defined by the following fields:

- **id:** Integer, Primary Key, Auto-Increment.
- **email:** String, Unique, Not Null.
- **password\_hash:** String, Not Null.
- **age\_group:** String, Nullable.
- **preferred\_language:** String, Not Null.
- **is\_admin:** Boolean, Default: False.

#### 3.2. User Registration and Hashing

The /register endpoint (POST) handles new user creation. Passwords are never stored in plaintext.

- **Hashing:** Passwords are processed through a strong hashing library (e.g., Werkzeug's security module) before being stored as `password_hash`.
- **Default Settings:** Default `preferred_language` is 'en' and `is_admin` is set to False.
- **Key Initialization:** The very first registered user is automatically designated as the initial administrator.


### 3.3. Secure Session Management (JWT)

Security is paramount, implemented using Flask-JWT-Extended with HTTP-only cookies.

- **Login (/login):** Upon verification, a JWT access token is generated, including `user_id`, `age_group`, and `preferred_language` in its payload.
- **Security:** The token is transmitted via an **HTTP-only cookie**, preventing client-side JavaScript access and mitigating Cross-Site Scripting (XSS) risks.
- **Protected Endpoints:** All critical endpoints are decorated with `@jwt_required(locations=["cookies"])`.

### 3.4. Page Access Control

- **Unauthorized Access:** Server-side logic redirects unauthenticated users attempting to access `/chat` or `/profile` to the `/login` page.
  - **Frontend Logic:** JavaScript modules handle token expiration and automatically redirect the client upon receiving a 401 Unauthorized response from a protected API call.
-




# WellBot

Your Global Wellness Assistant

[Sign In](#)[Sign Up](#)

@ Email Address

🔒 Password



☐ Remember me

[Forgot password?](#)

[Sign In](#)

By continuing, you agree to WellBot's [Terms of Service](#) and [Privacy Policy](#)

## 4. Milestone 2 – Core System: Chatbot Integration & Web UI

This milestone centered on building the **core conversational engine** and linking it to the web interface. It involved implementing the **Rasa NLU model** for intent recognition, designing the **structured HealthKnowledge database**, creating the Python utility for knowledge import, and developing the **Action Server logic** to query the database and retrieve relevant, non-diagnostic guidance. Finally, the basic chat UI was integrated with the Flask API.

### 4.1. Rasa NLU Model Implementation

The NLU model is trained to identify key intents:

- greet, goodbye, symptom\_inquiry, first\_aid\_request, wellness\_advice.

The Configuration (config.yml) utilizes a standard language-specific pipeline optimized for both English and Hindi text processing.

### 4.2. Structured Health Knowledge Base Design

The **HealthKnowledge** table stores the core intelligence of WellBot:

- id, intent\\_name, keyword, response\\_en, response\\_hi, age\\_group\\_filter.

### 4.3. Data Import Utility (load\_knowledge.py)

A dedicated Python script manages the ingestion of knowledge data. It reads from a standardized CSV, uses the **Pandas** library for data cleaning, and leverages **SQLAlchemy** to batch-insert or update entries in the HealthKnowledge table.

### 4.4. Action Server (actions.py) Logic

The custom action `action_query_knowledge_base` executes the business logic:

1. **Input:** Receives the intent and user metadata (language, age group).
2. **Query Construction:** Constructs a dynamic SQLAlchemy query.
3. **Filtering:** Filters the query by the user's preferred\_language and applies the age\_group\_filter.

4. **Output:** Returns the retrieved, localized guidance text back to Rasa Core.

## 4.5. Frontend Structure and Interactivity

The frontend is a lightweight, responsive interface served by Flask.

- **Core Templates:** login.html, register.html, profile.html, and chat.html.
  - **JavaScript Interactivity (chat.js):** Manages asynchronous message submission and handles the real-time display of bot responses.
-



## Complete Your Profile

### Name

### Age Group

### Gender



Male



Female



Other

### Exercise Hours (per day)

### Health Conditions (select all that apply)

☒ None☐ Diabetes☐ High BP☐ Thyroid☐ Asthma☐ Heart Disease☐ Obesity☐ Joint Pain☐ Stress

### Language Preference

☒ English☐ हिंदी

## 5. Milestone 3 – Expanded Features: Advanced NLP & Multilingual Capabilities

The focus here was on achieving **global reach and improved intelligence**. This involved expanding the knowledge base content, introducing **multilingual capabilities** (Hindi support) by updating Rasa pipelines and training data, implementing **dynamic language output** based on user profiles, and integrating **advanced Entity Extraction** (duration, severity, location) to enable more nuanced and personalized advice retrieval. Crucially, **safety disclaimers** were also enforced.

### 5.1. Knowledge Base Expansion Strategy

Content was expanded to over 100 entries, focusing on a preventative and holistic wellness approach.

#### New Content Categories:

- Common Illnesses and Symptom Management, Preventive Measures, Wellness Recommendations, Chronic Condition Management.

### 5.2. Safety and Compliance Disclaimers

- **Disclaimer Implementation:** An automated utter\_disclaimer action is triggered at the start of every new conversation session via a high-priority rule in rules.yml.
- **Content:** Explicitly states that WellBot provides **non-diagnostic guidance only**.

### 5.3. Multilingual Rasa Configuration

Hindi language support was introduced.

- **Configuration Update (config.yml):** The Rasa pipeline was updated to include language-agnostic components or separate models.
- **Training Data:** NLU examples were segregated into language-specific files (data/en/nlu.yml and data/hi/nlu.yml).

## 5.4. Dynamic Language Output Implementation

Response language is now determined by the user's stored profile preference.

- **Metadata Flow:** The Flask backend injects the user's preferred\_language into the Rasa tracker's slot.
- **Action Server Logic:** action\_query\_knowledge\_base uses this parameter to dynamically select the correct bilingual column from the HealthKnowledge table.

## 5.5. Advanced Entity Extraction and Slot Filling

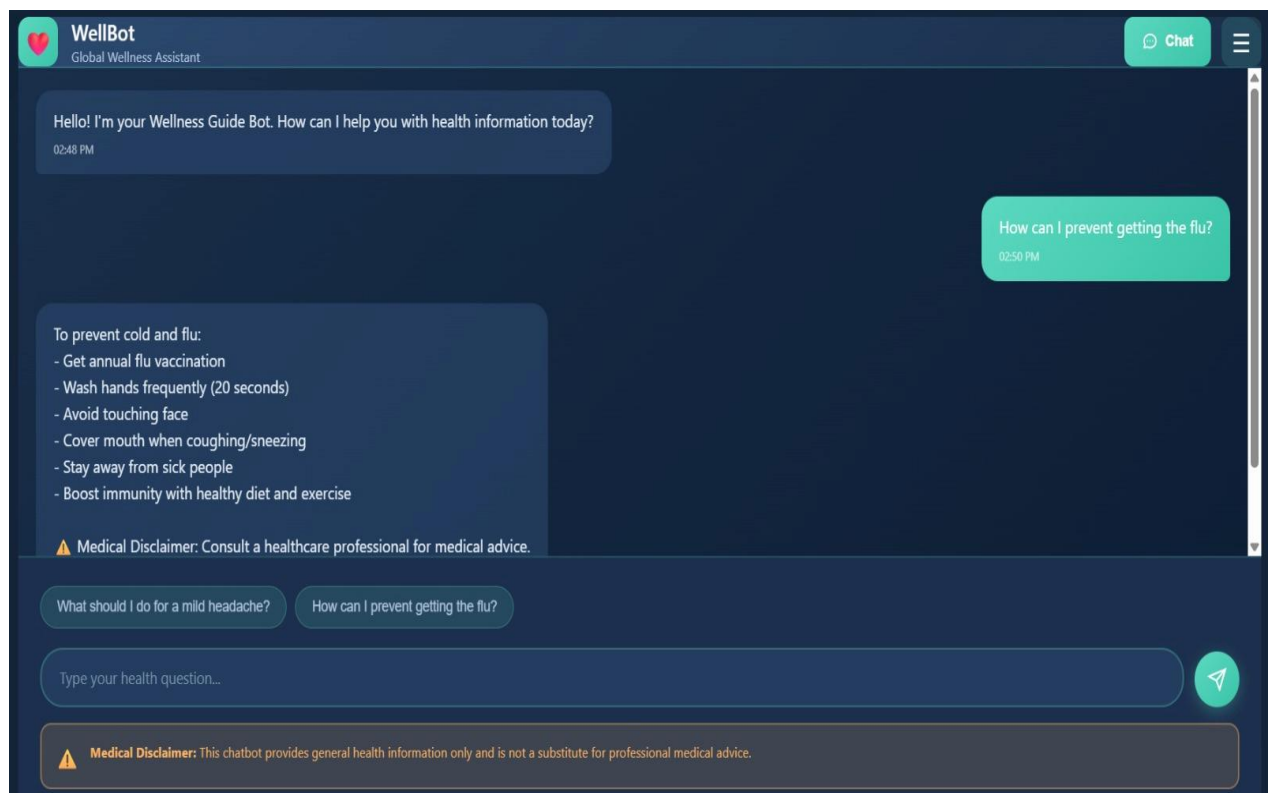
Specific entities were introduced to improve personalization:

- duration, severity, location.

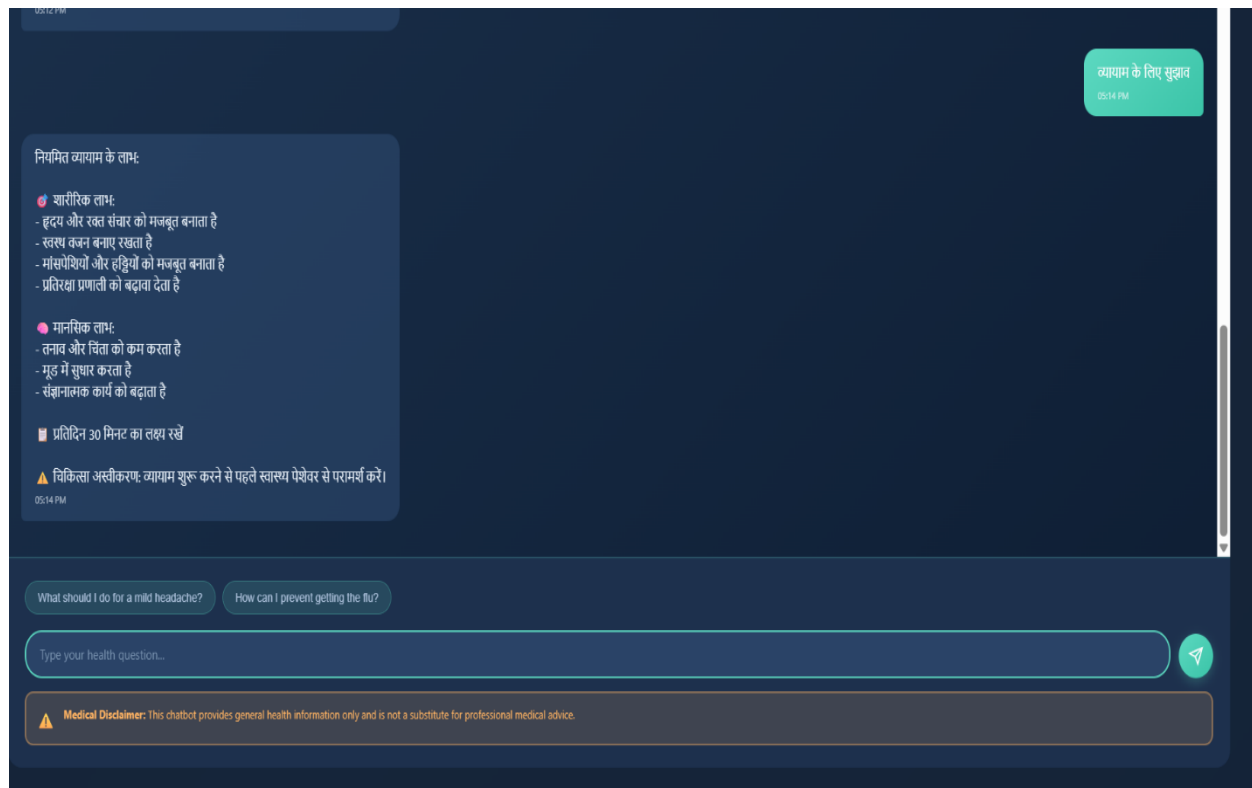
Training data was extensively updated in both English and Hindi to accurately recognize and extract these context variables.

---

## English Chats



## Hindi Chats



## 6. Milestone 4 – Admin Dashboard & System Optimization

This final milestone focused on **maintenance, quality control, and system hardening**. Key components included building a protected **Admin Dashboard** for administrators, implementing the **Knowledge Base Management Module** (allowing CRUD operations on health content), integrating a **User Feedback System** for continuous improvement, and developing an **Analytics and Reporting** module. System hardening focused on robust **database session management** and improved exception handling for reliability.

### 6.1. Admin Dashboard Features and Roles

The admin dashboard is a protected Flask Blueprint (/admin) accessible only to users with the `is_admin: True` flag.

- **Core Role:** Content management, user feedback review, and system monitoring.
- **Access Control:** Requires JWT authentication and an additional server-side check for `is_admin` status.

### 6.2. Knowledge Base Management Module

This module allows administrators to maintain the quality and relevance of WellBot's advice.

- **Functionality:** Administrators can **Browse/Search** all entries in the HealthKnowledge table. Web forms enable **Add/Edit** and secure **Delete** operations for bilingual content.

### 6.3. User Interaction Analytics and Reporting

A dedicated analytics module provides insights into system usage.

#### Key Metrics Displayed:

- Total User Interactions, Top 10 Most Common Queried Intents, Distribution of Users by Age Group and Preferred Language, Average Feedback Rating.

### 6.4. User Feedback System Design

A feedback mechanism (Thumbs-up/Thumbs-down) was integrated into the chat interface.

- **Data Model (ChatFeedback Table):** Captures feedback data including user\_id, knowledge\_id, rating, comment, and timestamp.
- **Endpoint:** The /feedback endpoint handles the submission.

## 6.5. System Hardening and Database Optimization

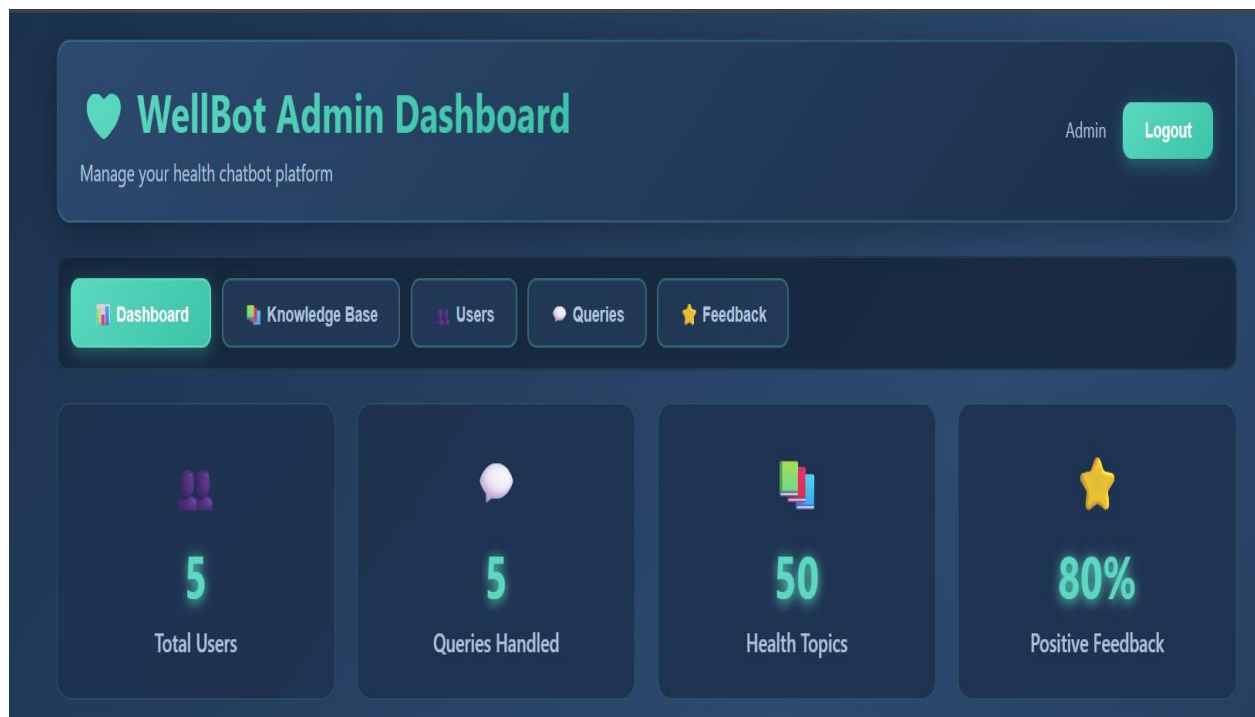
- **Database Session Management:** Implemented robust SQLAlchemy session handling with proper rollback and commit logic to prevent concurrent operation errors.
- **Administrative Setup:** The first user created is automatically granted is\_admin: True.
- **Exception Handling:** Enhanced error logging and graceful exception handling across Flask API endpoints.

## 6.6. UI/UX Modernization Summary

The frontend was refactored for improved aesthetics and usability, adopting a sleek design, responsive layout, and polished authentication flow.

---

## Dashboard




# Query Trends Over Time



### Knowledge Base:


## User, user query and Feedbacks:

 **WellBot Admin Dashboard**



Manage your health chatbot platform

Admin [Logout](#)

[Dashboard](#) [Knowledge Base](#) [Users](#) [Queries](#) [Feedback](#)

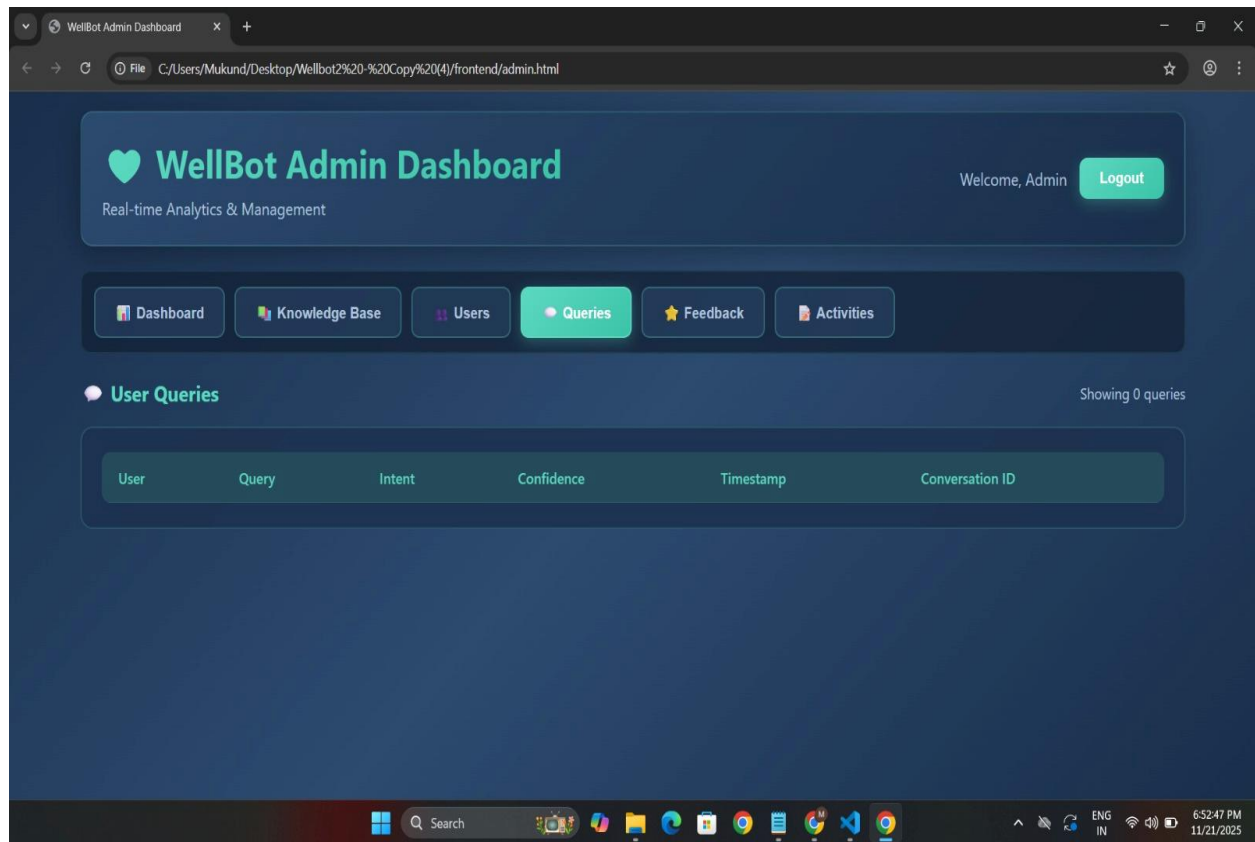
 **User Management**

Total: 5 users

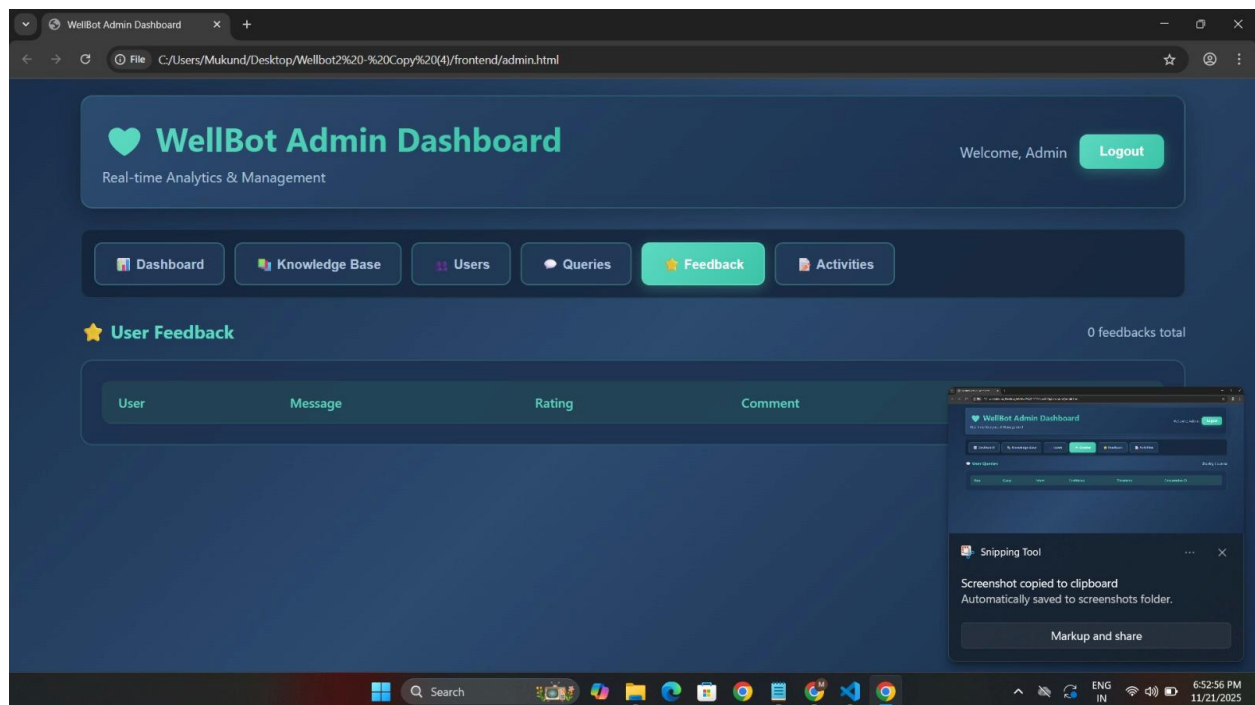
Username	Email	Age Group	Language	Conversations	Joined
Megha M	megha@gmail.com	18-25	English	12	 
Arati m	arati@gmail.com	18-25	Hindi	8	
Mukund Macharla	mukundmacharla@gmail.com	18-25	English	15	
Uma Macharla	umamacharla26@gmail.com	36-45	English	5	
Megha M	megha123@gmail.com	18-25	English	2	

## Recent User Queries

r



## User Feedback & Ratings



## Database schema:

### WellBot Database Schema Description

The database schema is organized around a central USERS table, linking out to logs and data specific to each user. The knowledge base is modularized into Q&A and Tips tables.

---

## User and Core Tables

### 1. USERS (Primary Table)

This is the main authentication and profile table.

- **PK (Primary Key):** id (user)
- session\\_id
- created\\_at
- language\\_preference
- **Relationship:** Has a **1-to-many** relationship with CONVERSATIONS and EXERCISE LOGS.

### 2. CONVERSATIONS

Logs individual message exchanges for analytics and dialogue history.

- **PK (Primary Key):** conversation\\_id
- message (text)
- sender (FK from USERS)
- timestamp
- language (for multilingual analysis)
- **Relationship:** Linked back to USERS with a **1-to-many** relationship (one user can have many conversations).

### 3. EXERCISE LOGS

Tracks user-reported exercise activity.

- **PK (Primary Key):** id (integer)
  - **FK (Foreign Key):** user\\_id (links to USERS table)
  - exercise\\_type
  - exercise\\_date
  - sender (integer, FK)
  - timestamp
  - **Relationship:** Linked back to USERS with a **1-to-many** relationship (one user can have many exercise logs).
-

## Health Data and Knowledge Base Tables

### 4. USER HEALTH DATA

Stores daily or periodic health metrics logged by the user.

- **PK (Primary Key):** user\\_h\\_id (NOT NULL)
- **FK (Foreign Key):** user\\_b\\_id (links to USERS table)
- water\\_intake
- current\\_mood
- sleep\\_hours
- date
- **Relationship:** Has a **1-to-many** relationship with HEALTH QA (though the relationship flow on the diagram suggests that HEALTH QA links to USER HEALTH DATA via user\\_h\\_id which seems non-standard for Q&A, usually Q&A is static).

### 5. HEALTH QA (Question & Answer)

The primary knowledge base for specific health inquiries.

- **PK (Primary Key):** HEALTH\\_QA
- question
- answer
- category
- authority\\_references
- **Relationship:** Shares a **1-to-many** relationship with WELLNESS TIPS.

### 6. WELLNESS TIPS

Stores general wellness advice, potentially linked to specific health categories.

- **PK (Primary Key):** TIP

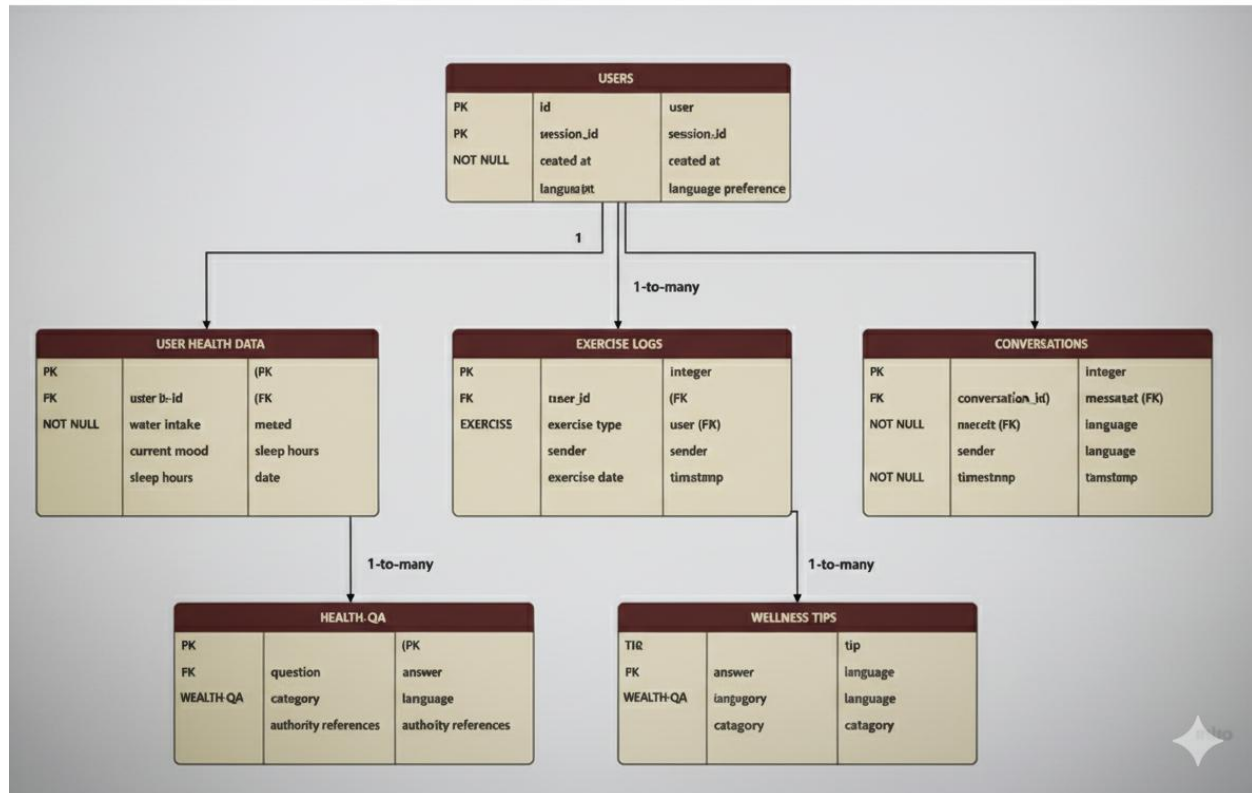
- **FK (Foreign Key):** WEALTH\\_QA (links to HEALTH QA table)
- tip (the advice text)
- language
- category
- **Relationship:** Linked back to HEALTH QA with a **1-to-many** relationship.

## 7. CONVERSATIONS (Secondary Entry)

This seems to be a redundant or specialized logging table for message segments.

- **PK (Primary Key):** integer (likely message\\_id)
- **FK (Foreign Key):** conversation\\_id (links to the main CONVERSATIONS table)
- message\\_segment
- language
- timestamp
- **Note:** The structure suggests it might be intended for message fragmentation or slot tracking, though its definition in the diagram is slightly ambiguous compared to the main CONVERSATIONS table.

## WellBot Database Schema Diagram



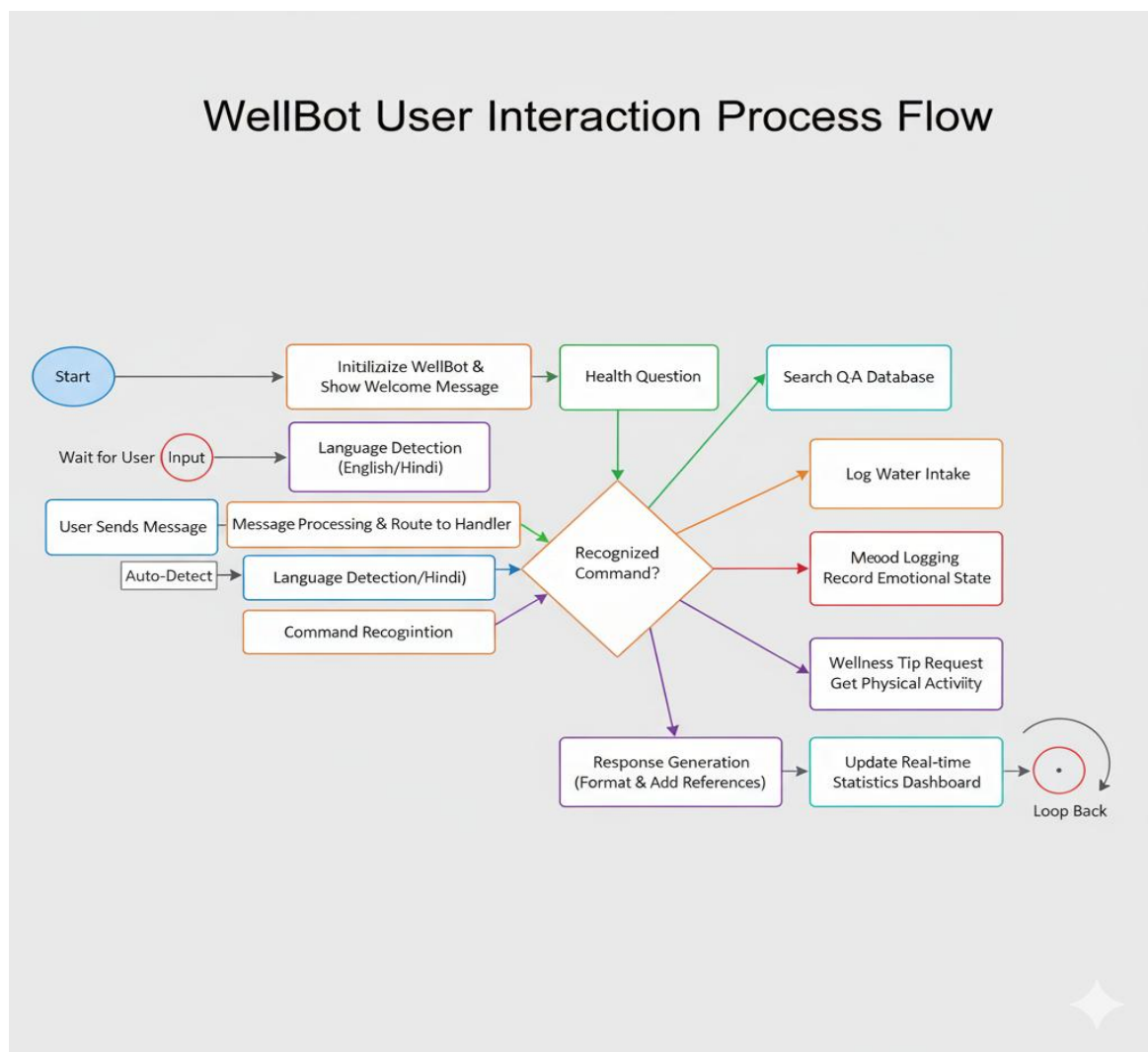
## Diagrams:

### User Flow diagram:

**Purpose:** Visualizes the **path a user takes** through a website or application to complete a specific goal. It's focused on the user experience (UX).

**What it shows:** A series of steps, screens (or pages), and **decision points** (e.g., a diamond shape for "Yes/No") from the entry point to the final desired action (conversion).

**Analogy:** A road map or flowchart for a customer's journey from finding a product online to completing the purchase checkout.



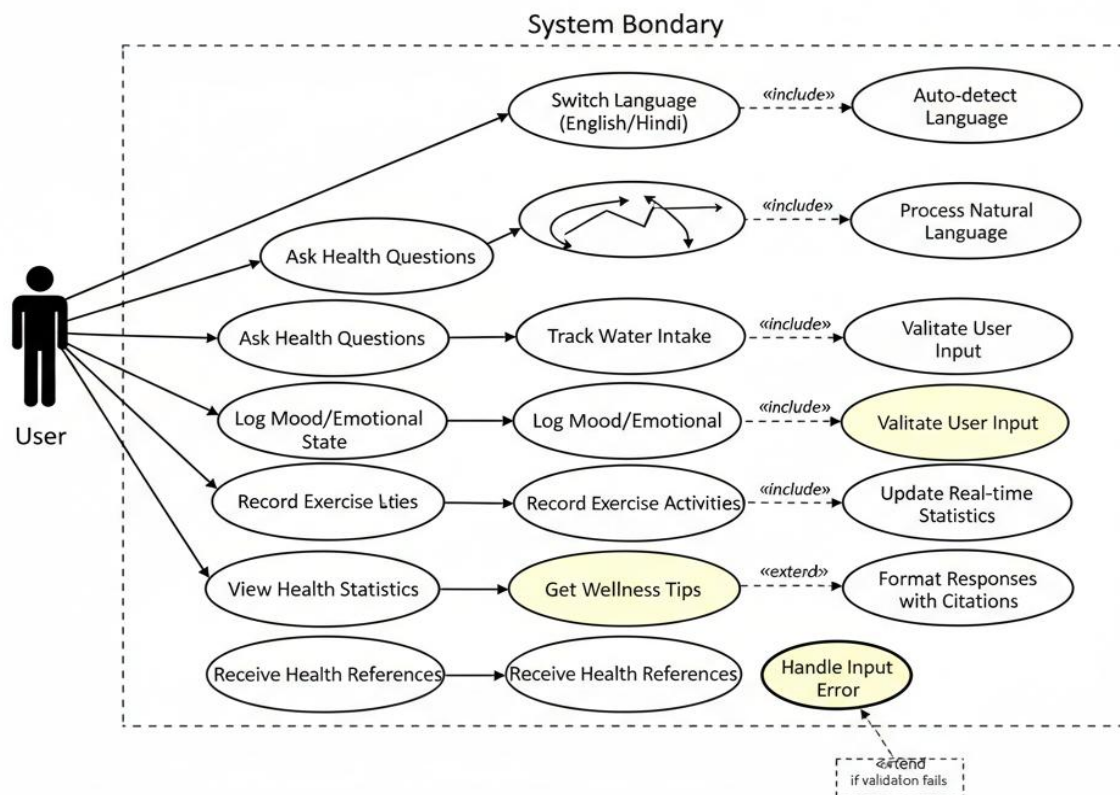
## Use case diagram:

**Purpose:** Captures the **functional requirements** and high-level scope of a system from the **external user's perspective**.

**What it shows:** The **actors** (external users or other systems) and the primary **use cases** (the discrete functions/goals the system provides, shown as ovals), linked by association lines.

**Analogy:** A menu for a restaurant, listing everything the customer (actor) can order (use case).

Diagram 3: UML Use Case Diagram  
Wellness Assistant System



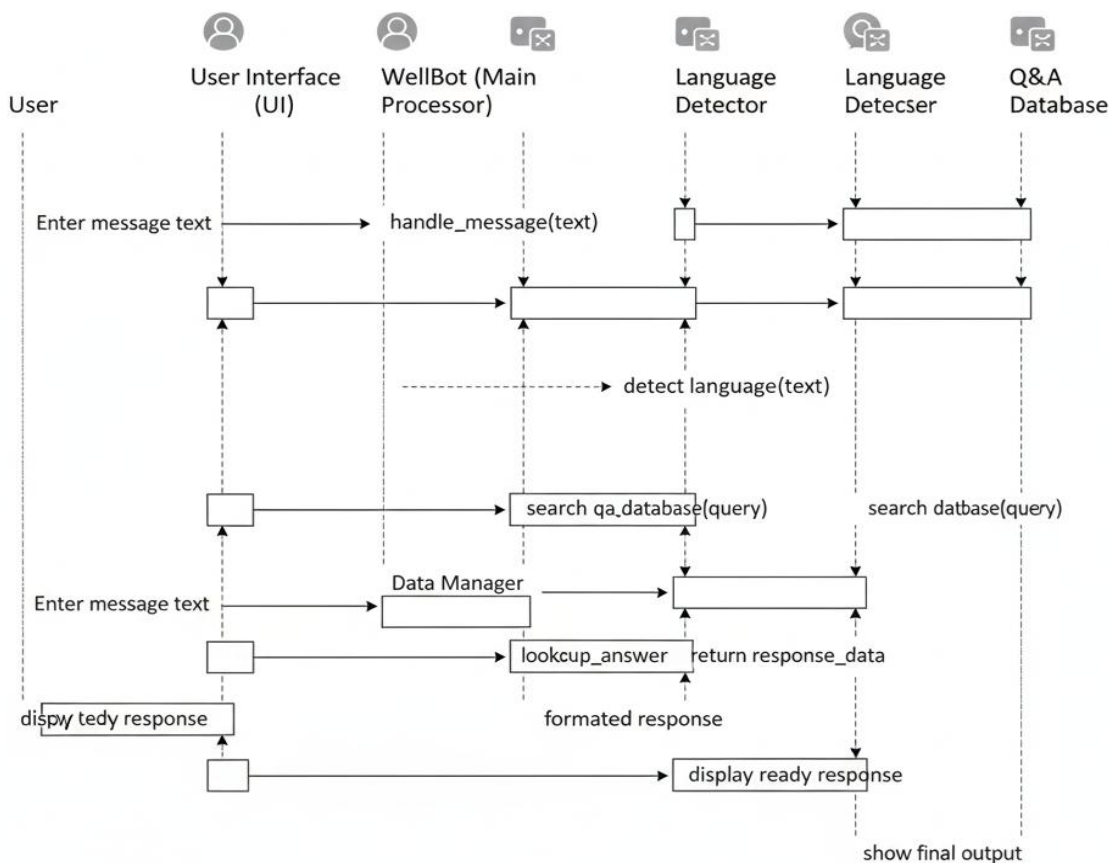
## Sequence Diagram:

**Purpose:** Shows the **order of interactions** between objects or components in a system over time to complete a specific task or scenario.

**What it shows:** **Lifelines** (vertical lines representing objects/components) and **messages** (horizontal arrows) exchanged in the exact chronological sequence they occur. It focuses on the time-ordered logic.

**Analogy:** A script for a play, showing exactly which character (object) speaks (sends a message) to which other character and when.

Diagram 4: UML Sequence Diagram  
Message Processing Workflow



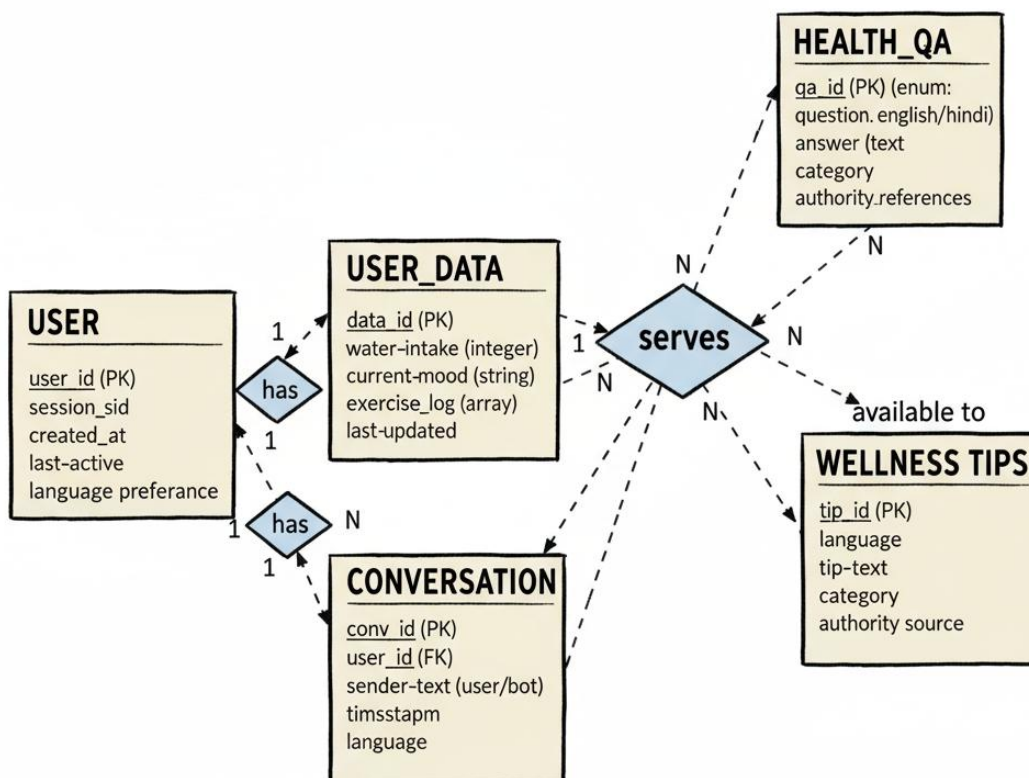
## Entity Relation diagram:

**Purpose:** Models the **structure of a database**.

**What it shows:** The primary **entities** (tables or concepts) in a system, their **attributes** (columns), and the **relationships** between them, often detailing **cardinality** (one-to-one, one-to-many, etc.).

**Analogy:** A schematic showing all the key data points in a library (books, members, loans) and how they are linked.

Diagram 5: Entity Relationship Diagram  
Chatbot Data Structure



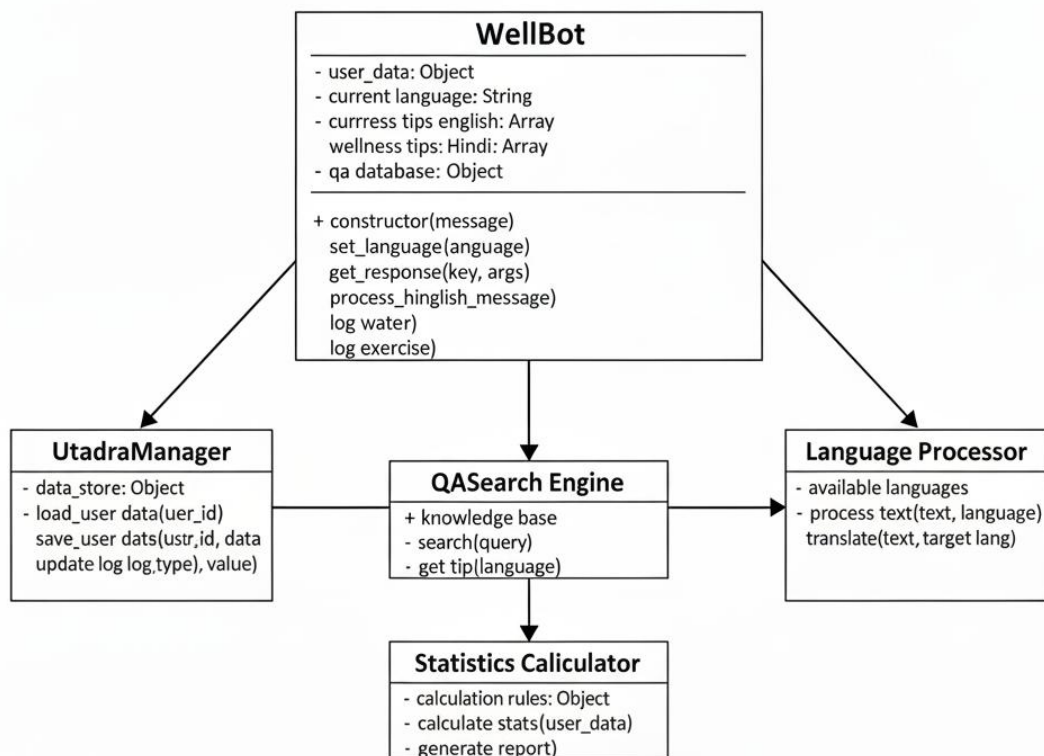
## Class Diagram:

**Purpose:** Describes the **static structure** of an object-oriented system. It's the blueprint for the code.

**What it shows:** The system's **classes**, their **attributes** (data), **operations/methods** (functions), and the **relationships** (like inheritance, association, or aggregation) between them.

**Analogy:** The architectural blueprint of a house, detailing every component (room, wall, window) and how they are structurally connected.

## WellBot System Class Diagram



## 7. Conclusion and Future Work

WellBot successfully delivers a secure, multilingual, and data-driven wellness assistant built on a robust Flask/Rasa architecture. By integrating a customizable knowledge base with advanced NLP capabilities, it fulfills its primary goal of providing accessible, non-diagnostic guidance in both English and Hindi, all within a professional and responsive web application. The implementation of authentication, the Admin Dashboard, and the feedback system ensures the project is not only functional but also secure, maintainable, and iteratively scalable.

### 7.2. Recommended Future Enhancements

- **Integration:** Migrate from SQLite to a cloud-based database (e.g., **PostgreSQL**, **Firestore**) for global deployment and improved concurrency/scalability.
- **Conversational Flow:** Implement Forms/Flows for structured information gathering (e.g., asking about symptom duration, pain level) to enhance the conversational depth.
- **Language:** Addition of **Spanish and French** language support.
- **Personalization:** More extensive use of age\_group and other profile data to customize advice more precisely.