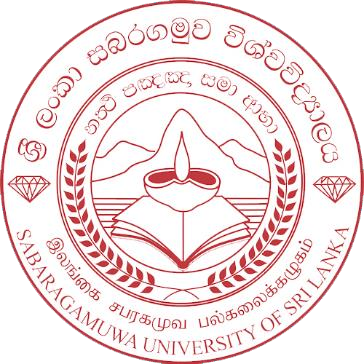
SMART HEALTH ASSISTANT: A PERSONALIZED HEALTH ADVISORY SYSTEM



Indika Balasuriya

Index Number

Master of Information Technology

Faculty of Graduate Studies Sabaragamuwa University of Sri Lanka

Month 2025

# Declaration

I declare that this report does not incorporate, without acknowledgment, any material previously submitted for a Degree or a Diploma in any University and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text.

Also, I hereby grant to Sabaragamuwa University of Sri Lanka the non- exclusive right to reproduce and distribute my report, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Name of Student Signature of Student Date

# Certificate of Approval

We hereby declare that this report is from the student’s own work and effort, and all other sources of information used have been acknowledged. This report has been submitted with our approval.

|  |  |  |
| --- | --- | --- |
| Coordinator | Signature | Date: |
| Internal Supervisor | Signature | Date: |

# Acknowledgement

I would like to express my heartfelt gratitude to all those who contributed to the successful completion of this project proposal.

First and foremost, I extend my deepest appreciation to my internal supervisor, [Supervisor's Name], whose invaluable guidance, constructive feedback, and continuous encouragement inspired me throughout this project. Their expertise and critical insights were instrumental in shaping the direction and quality of my work.

I am also profoundly thankful to the course coordinator and the academic staff of the Faculty of Graduate Studies, Sabaragamuwa University of Sri Lanka, for providing an excellent academic environment, timely advice, and the necessary resources that greatly supported my efforts.

I would like to acknowledge the support of my friends and colleagues, who offered motivation and shared valuable knowledge during difficult moments. Their unwavering support made this journey significantly smoother.

A special word of thanks goes to my family for their constant encouragement, patience, and unconditional love, which empowered me to pursue my studies with dedication and confidence.

Finally, I extend my sincere thanks to all individuals and organizations whose contributions, directly or indirectly, facilitated the realization of this project.

# Abstract

The Smart Health Assistant is a mobile and web-based health advisory system designed to provide personalized, preventive healthcare recommendations using Artificial Intelligence. By collecting user-specific parameters such as age, gender, and self-reported symptoms, the system generates tailored health screenings, nutrition guidance, and exercise plans. A key feature is the integration of a GPT-powered AI chatbot that enables natural language interaction to analyze symptoms and suggest relevant diagnostic tests. The application also delivers email reminders to encourage regular checkups and adherence to wellness plans. Developed using Python, Streamlit, OpenAI API, and SQLite, the system ensures data security, user-friendly design, and scalability. The project promotes early detection, reduces dependence on static health information, and contributes to the broader goal of making healthcare accessible and proactive. The system’s architecture emphasizes modularity and responsiveness across platforms, enabling future enhancements such as wearable integration. Ultimately, the Smart Health Assistant demonstrates how AI-driven tools can empower individuals to take ownership of their health, improving outcomes and aligning with global digital health strategies.

**Table of Contents**

[Declaration 2](#_Toc211525685)

[Certificate of Approval 3](#_Toc211525686)

[Acknowledgement 4](#_Toc211525687)

[Abstract 5](#_Toc211525688)

[Chapter 1 Introduction 11](#_Toc211525689)

[1.1 Background 11](#_Toc211525690)

[1.2 Problem Statement 11](#_Toc211525691)

[1.3 Aim 12](#_Toc211525692)

[1.4 Scope 12](#_Toc211525693)

[1.5 Significance 13](#_Toc211525694)

[Chapter 2 Objectives 14](#_Toc211525695)

[2.1 Main Objective 14](#_Toc211525696)

[2.2 Specific Objectives 14](#_Toc211525697)

[2.2.1 Develop a User Registration and Authentication Module 14](#_Toc211525698)

[2.2.2 Design and Implement a Personalized Health Recommendation Engine 14](#_Toc211525699)

[2.2.3 Integrate a Symptom Checker Chatbot 15](#_Toc211525700)

[2.2.4 Implement a Notification and Alert System 15](#_Toc211525701)

[2.2.5 Ensure Secure Data Management 15](#_Toc211525702)

[2.2.6 Design a User-Centric Interface 16](#_Toc211525703)

[2.2.7 Perform Testing and Validation 16](#_Toc211525704)

[Chapter 3 Proposed System 17](#_Toc211525705)

[3.1 Overview of the Proposed System 17](#_Toc211525706)

[3.2 System Architecture 17](#_Toc211525707)

[3.3 Key Functional Modules 19](#_Toc211525708)

[3.3.1 User Registration and Profile Management 19](#_Toc211525709)

[3.3.2 Personalized Health Plan Generator 19](#_Toc211525710)

[3.3.3 Email Reminders 19](#_Toc211525711)

[3.4 User Interaction Flow 19](#_Toc211525712)

[Chapter 4 Functional and Non-Functional Requirements 21](#_Toc211525713)

[4.1 Functional Requirements 21](#_Toc211525714)

[4.2 Non-Functional Requirements 22](#_Toc211525715)

[4.2.1 Performance 22](#_Toc211525716)

[4.2.2 Scalability 23](#_Toc211525717)

[4.2.3 Reliability 23](#_Toc211525718)

[4.2.4 Security 23](#_Toc211525719)

[4.2.5 Usability 23](#_Toc211525720)

[4.2.6 Compatibility 23](#_Toc211525721)

[4.2.7 Maintainability 24](#_Toc211525722)

[Chapter 5 Technologies and Resources 25](#_Toc211525723)

[5.1 Technologies 25](#_Toc211525724)

[5.1.1 Python and Streamlit – The Application Core 25](#_Toc211525725)

[5.1.2 OpenAI GPT API (Chatbot AI Engine) 26](#_Toc211525726)

[5.1.3 SQLite and bcrypt: Data Persistence and Security 26](#_Toc211525727)

[5.1.4 Cloud Hosting (AWS/Google Cloud/Heroku) 27](#_Toc211525728)

[5.2 Implementation Details 28](#_Toc211525729)

[5.3 Innovative Implementation of Algorithms and Data Structures 28](#_Toc211525730)

[5.5.1 Rule-Based Expert System for Health Recommendations 28](#_Toc211525731)

[5.5.2 State Management in a Stateless Framework (Streamlit) 28](#_Toc211525732)

[5.5.3 Rule-Based Expert System for Health Recommendations 29](#_Toc211525733)

[5.5.4 Rule-Based Expert System for Health Recommendations 31](#_Toc211525734)

[Chapter 6: System Architecture 33](#_Toc211525735)

[6.1 Overview of System Architecture 33](#_Toc211525736)

[6.2 Architectural Components and Implementation 34](#_Toc211525737)

[6.2.1 User Interface (UI) and Backend Server 34](#_Toc211525738)

[6.2.2 AI Chatbot Service Integration 34](#_Toc211525739)

[6.2.3 The Recommendation Engine as a Rule-Based System 34](#_Toc211525740)

[6.2.4 Authentication and Password Management 35](#_Toc211525741)

[6.2.5 Authentication and Password Management 35](#_Toc211525742)

[6.2.6 Authentication and Password Management 35](#_Toc211525743)

[6.3 System Architecture Diagram 36](#_Toc211525744)

[6.4 Use Case Diagram 36](#_Toc211525745)

[6.5 Data Flow Diagram (DFD) — Level 0 37](#_Toc211525746)

[6.6 ER Diagram 38](#_Toc211525747)

[Chapter 7: Results and Evaluation 39](#_Toc211525748)

[7.1 Demonstration of System Functionality: A Detailed Walkthrough 39](#_Toc211525749)

[7.1 Interrelationship of Experimental Results: A Cohesive User Journey 42](#_Toc211525750)

[7.3 Analysis of Achieved Accuracy 42](#_Toc211525751)

[7.4 Implications and Limitations 44](#_Toc211525752)

[Chapter 8: Future Work 45](#_Toc211525753)

[8.1 Gaps of the Current Project 46](#_Toc211525754)

[8.1 Proposals for Enhancement and Re-Design: Charting the Path Forward 47](#_Toc211525755)

[8.3 Technical Implementation Proposal 47](#_Toc211525756)

[8.2 Evolving the Recommendation Engine into an Adaptive Learning System 48](#_Toc211525757)

[8.4 Architecting a Stateful, Context-Aware AI Health Companion 48](#_Toc211525758)

[8.5 Building a Production-Grade, Secure Infrastructure 49](#_Toc211525759)

[8.6 Expanding into Specialized Health Modules and Gamification 49](#_Toc211525760)

[Chapter 9: Project Timeline 50](#_Toc211525761)

[8.1 Overview 50](#_Toc211525762)

[8.2 Gantt Chart 51](#_Toc211525763)

[Chapter 9: Conclusion 53](#_Toc211525764)

[9.1 Conclusion 53](#_Toc211525765)

[Reference 54](#_Toc211525766)

**Table of figures**

[Figure 1 System Architecture Diagram 18](#_Toc211525499)

[Figure 2 User Interaction Diagram 20](#_Toc211525500)

[Figure 3 Secure Password Hashing registration 31](#_Toc211525501)

[Figure 4 Registration 31](#_Toc211525502)

[Figure 5 Secure Password Hashing login 32](#_Toc211525503)

[Figure 6 Login Page 32](#_Toc211525504)

[Figure 7 AI Chatbot API Call 32](#_Toc211525505)

[Figure 8 Authentication Gate 33](#_Toc211525506)

[Figure 9 Use Case Diagram for Smart Health Assistant 36](#_Toc211525507)

[Figure 10 DFD diagram 37](#_Toc211525508)

[Figure 11 ER Diagram 38](#_Toc211525509)

[Figure 12 Gantt chart 51](#_Toc211525510)

**Table of tables**

[Table 1 System architecture 17](#_Toc211524659)

[Table 2 Core features critical to achieving the system’s goals 21](#_Toc211524660)

[Table 3 Hardware Requirements 27](#_Toc211524661)

[Table 4 Software Requirements 27](#_Toc211524662)

[Table 5 Challenge: Over-Ambitious Project Aims vs. Practicality 29](#_Toc211524663)

[Table 6 Challenge: Difficulties with Existing Software (Streamlit's Execution Model) 30](#_Toc211524664)

[Table 7Challenge: Ensuring Responsible AI Interaction 30](#_Toc211524665)

# Chapter 1 Introduction

## Background

The evolution of information technology has fundamentally reshaped the landscape of the global healthcare sector, introducing new paradigms for patient care and wellness management. A significant development in this transformation is the rise of mobile health, commonly known as mHealth, which utilizes mobile and web technologies to support health objectives. These applications are becoming increasingly instrumental in the domains of disease prevention, continuous health monitoring, and the delivery of healthcare services tailored to individual needs. In contemporary digital society, there is a pronounced demand for healthcare solutions that are not only immediately accessible but also deeply personalized and focused on preventive measures, freeing individuals from the traditional constraints of geographical location and time. The Smart Health Assistant project is situated within this context, aiming to harness the power of artificial intelligence and web technologies to deliver a sophisticated, user centric health advisory service. It moves beyond generic information dissemination to provide actionable, personalized health intelligence directly to the user.

According to Statista (2024), the number of mobile health app users worldwide surpassed 1 billion, highlighting a growing reliance on digital healthcare solutions. Artificial Intelligence (AI), particularly Natural Language Processing (NLP), has introduced intelligent agents capable of providing real-time health advice, preliminary diagnostics, and preventive care recommendations (Topol, 2019).

## Problem Statement

Current healthcare delivery models, even those incorporating digital tools, often offer generic recommendations without accounting for individual characteristics. As a result, users receive broad advice that may not adequately address their personal risks, leading to late detection of illnesses and suboptimal health outcomes (Patel et al., 2022).

Many contemporary digital health platforms and applications, while beneficial, often dispense generic health recommendations that do not fully account for the unique physiological and demographic characteristics of each user. This one size fits all approach can result in advice that is not optimally effective or relevant to an individual's specific health risks and conditions. Such broad guidance may contribute to the delayed detection of potential health issues and, consequently, lead to suboptimal health outcomes. Furthermore, a gap exists in the market for integrated systems where users can interact conversationally about their symptoms to receive immediate, intelligent feedback. Many existing symptom checkers lack the sophisticated natural language understanding capabilities required to guide users toward appropriate preliminary diagnostic actions effectively. There is therefore a critical need for an intelligent system capable of analyzing user specific inputs in a dynamic, conversational manner to deliver personalized, actionable health recommendations on a scalable basis.

## Aim

The principal aim of this project is to conceptualize, design, and implement the Smart Health Assistant. This system is envisioned as a web based personalized health advisory platform that intelligently leverages artificial intelligence and user provided information. Its core purpose is to generate highly tailored health recommendations, perform preliminary symptom analyses, and deliver timely preventive health alerts to its users, thereby fostering a proactive approach to personal wellness.

## Scope

The Smart Health Assistant application will offer the following functionalities:

The Smart Health Assistant application is designed to provide a comprehensive suite of functionalities to promote proactive health management. The system will support both registered users and guest users. A full user registration module will be available, allowing individuals to create persistent profiles. For those preferring temporary access, a guest mode is implemented, which facilitates immediate use of core features without requiring permanent data storage.

A central feature of the system is its capacity to generate personalized health screenings, exercise plans, vitamin suggestions, and meal plans. These recommendations are algorithmically tailored based on fundamental user demographics, specifically age and gender, to ensure relevance. The application is specifically targeted at adults and stratifies its advice across distinct age categories: 20 to 30, 30 to 40, 40 to 50, 50 to 60, and 60 years and older.

To provide interactive diagnostic support, the system incorporates a symptom checker chatbot. This module utilizes an advanced artificial intelligence model, specifically the OpenAI GPT model, to interpret user described symptoms in natural language. Based on this input, the chatbot suggests potential diagnostic tests and offers preliminary evaluations, serving as a first point of inquiry.

The application also includes a notification system to encourage consistent engagement with health monitoring activities. Registered users will receive email reminders for important health actions, such as regular checkups or screenings. To ensure data integrity and user privacy, the system will securely store user health profiles, the history of chatbot interactions, and all generated health recommendations in a structured database.

## Significance

The Smart Health Assistant is poised to make a substantial contribution to public health by empowering individuals to assume a more active and informed role in managing their own health. By providing accessible and personalized guidance, the system encourages preventive healthcare practices, which are crucial for the early detection of potential health issues and the overall reduction of the burden of non-communicable diseases.

Furthermore, the integration of AI-driven chatbots ensures a scalable and efficient means of interacting with users, minimizing the dependency on physical healthcare visits for preliminary advice. This aligns with global trends toward telemedicine and self-managed healthcare (Lee & Yoon, 2021).

The integration of an AI powered chatbot offers a scalable and efficient method for delivering preliminary health advice, thereby reducing the immediate dependency on clinical consultations for initial guidance. This aligns perfectly with the growing global trends toward telemedicine and self managed healthcare, making health advisory services more convenient and widely available. From a technological standpoint, this project serves as a practical demonstration of how emerging technologies, including machine learning, web computing, and API services, can be synergistically combined to create innovative solutions that enhance the accessibility, personalization, and effectiveness of healthcare delivery.

# Chapter 2 Objectives

## Main Objective

The main objective of this project is to develop a comprehensive Smart Health Assistant system. This system will function as a personalized health advisory service delivered through a dynamic web platform, with its intelligence driven by advanced Artificial Intelligence technologies. The system is designed to meticulously analyze user specific parameters, including age, gender, and self-reported symptoms, to generate customized health recommendations, suggest pertinent diagnostic tests, and offer valuable preventive healthcare advice, all with the overarching goal of enhancing individual well-being and promoting long term health.

## Specific Objectives

To realize the main objective, the project is broken down into several specific, measurable objectives that guide its development and implementation.

## Develop a User Registration and Authentication Module

A primary objective is to build a secure and reliable user management system. This involves implementing functionality that allows users to create personal accounts by providing essential details such as a username, email, and password. The system design ensures that all user credentials, particularly passwords, are not stored in plaintext. Instead, the “bcrypt” library is utilized to perform cryptographic hashing, a critical security measure that protects user accounts from unauthorized access. The implementation also includes an option for guest mode, which grants temporary access to specific features of the application without the need for a formal registration process, thereby lowering the barrier to entry for new users.

## Design and Implement a Personalized Health Recommendation Engine

Another core objective is the creation of a sophisticated recommendation engine. This engine is responsible for generating health plans that are precisely tailored to the individual. The logic, encapsulated within the “health\_recommendations.py” file, uses the user’s demographic data, primarily age and gender, as inputs. Based on these parameters, the system references a predefined set of health guidelines to suggest appropriate preventive screenings, structured exercise regimens, specific nutritional advice, and beneficial vitamin supplements. To enhance the accuracy of these recommendations, users are categorized into relevant age brackets, ensuring that the guidance provided is appropriate for their specific life stage.

## Integrate a Symptom Checker Chatbot

This objective focuses on integrating an intelligent, conversational agent into the system. The symptom checker is powered by leveraging Natural Language Processing techniques through the OpenAI GPT series of models. The implementation, found in symptom\_chatbot.py, involves making secure API calls to the OpenAI service. The chatbot is designed to interpret a user's description of their symptoms, provided in conversational language, and in response, recommend suitable laboratory tests or suggest potential preliminary diagnoses. This feature acts as an initial, informative guide for users seeking to understand their health concerns.

## Implement a Notification and Alert System

To encourage proactive health management, the project includes the objective of implementing a robust notification system. This system is designed to send automated email alerts to registered users. These notifications serve as timely reminders for important health activities, such as upcoming annual screenings, recommended vaccination schedules, or periodic health checkups. The technical implementation, located in notifications.py, utilizes the Simple Mail Transfer Protocol (SMTP) to dispatch emails, ensuring reliable delivery of these crucial reminders.

## Ensure Secure Data Management

A critical objective is to ensure the secure and persistent storage of all user related data. The system utilizes a SQLite database, as defined in db.py, to manage this. All user interactions, including detailed health profiles, the complete history of recommendations provided, and logs of chatbot conversations, are stored within this database. Security is paramount; therefore, robust mechanisms for data handling and encryption of sensitive information like passwords are implemented to maintain user confidentiality and comply with standard data protection principles.

## Design a User-Centric Interface

The development of a clean, intuitive, and responsive user interface is a key objective to ensure the application is accessible and easy to use. The web interface is built using the Streamlit framework, which allows for the rapid development of interactive and data centric applications. The design prioritizes usability and an intuitive user experience, making it navigable for individuals across different age groups and with varying levels of technical proficiency. The interface is designed to be responsive, ensuring a consistent and effective user experience across different devices and screen sizes.

## Perform Testing and Validation

The final specific objective is to conduct thorough testing and validation of the entire system. This process involves rigorous usability testing to evaluate the effectiveness of the application, the ease of navigation through its features, and the overall user satisfaction. Furthermore, the accuracy of the health recommendations generated by the engine will be validated against established medical guidelines. Feedback will be systematically collected from test users to identify areas for improvement, allowing for the iterative refinement and optimization of the application’s performance and functionality before its final deployment.

# Chapter 3 Proposed System

## Overview of the Proposed System

The Smart Health Assistant is conceived as an intelligent health advisory platform, accessible via a web interface, designed to deliver highly personalized healthcare recommendations to its users. It represents a significant advancement over traditional, static health applications by incorporating a dynamic and interactive approach. At the heart of the system is a conversational chatbot, which allows users to engage in a natural dialogue to discuss their symptoms and health concerns. This interactive capability makes the healthcare advice provided by the system exceptionally responsive, deeply personalized, and broadly accessible to a wide audience, empowering users to take a more proactive stance toward their personal health and wellness.

## System Architecture

The architecture of the proposed system is built upon a modular design, comprising several key components that work in concert to deliver a seamless user experience. The primary components include a frontend web application, a backend application server, an AI chatbot engine, a persistent database, and a notification system.

Table System architecture

|  |  |
| --- | --- |
| **Component** | **Description** |
| **Frontend (Web App)** | Built using Streamlit and the frontend captures user input and displays personalized recommendations. |
| **Backend Server** | Python-based server handling data processing, user authentication, and communication with AI services. |
| **AI Chatbot Engine** | Integrated via OpenAI’s GPT API to process natural language symptom descriptions and generate diagnostic suggestions. |
| **Database (SQLite)** | Stores user profiles, health history, chatbot interactions, and recommendation logs securely. |
| **Notification System** | Email alerts about health checkups, preventive tests, and reminders. |
| **Cloud Hosting Platform** | AWS, Google Cloud, or Heroku to deploy the backend API and manage storage scalability. |

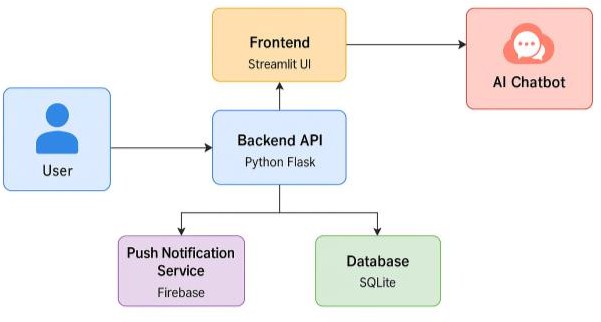


Figure System Architecture Diagram

The Frontend is the user facing component of the system. It is developed using the Streamlit framework, which excels at creating interactive and data driven web interfaces with Python. This layer is responsible for capturing all user inputs, such as registration details, profile information, and symptom descriptions, and for elegantly displaying the personalized health recommendations generated by the backend.

The Backend Server constitutes the core logic of the application. The provided implementation uses Python, with Streamlit handling the server-side processing. It manages critical functions such as user authentication, data processing, and orchestrating communication between the frontend, the database, and the external AI service.

The AI Chatbot Engine provides the system's intelligence. It is integrated via an API call to OpenAI's GPT service. This engine is tasked with processing the natural language descriptions of symptoms provided by users and generating relevant suggestions for diagnostic tests or preliminary evaluations based on its vast knowledge base.

The Database is the system's persistent storage layer. A SQLite database is used to securely store all essential data, including user profiles, detailed health histories, logs of all chatbot interactions, and the history of recommendations provided to each user.

Finally, the Notification System is responsible for proactive user engagement. It is implemented to send email alerts to users, reminding them of important health checkups, preventive tests, and other wellness related activities, thereby promoting consistent health monitoring.

## Key Functional Modules

The system's functionality is organized into distinct, well defined modules, each addressing a specific aspect of the user's health advisory journey.

## User Registration and Profile Management

This module forms the foundation of the user's interaction with the system. It allows individuals to register and create a secure, personal profile by providing key details such as their age, gender, and a basic health history. The system also accommodates a guest mode, which provides a way for users to access core functionalities, like the symptom checker, without the need for creating a permanent account and storing their personal data.

## Personalized Health Plan Generator

This is one of the most critical modules of the Smart Health Assistant. It is responsible for generating personalized health recommendations tailored to each registered user. The underlying algorithm processes the user's age group and gender to produce a comprehensive health plan. This plan includes specific suggestions for medical screenings, dietary supplements, a structured exercise regimen, and detailed meal plans, all designed to be relevant and actionable for the user.

## Email Reminders

This module focuses on ensuring that users remain engaged and proactive about their health. It is configured to send regular alerts and reminders via email. These notifications can inform users about upcoming vaccinations, necessary health screenings, or routine medical checkups. The system is designed to allow for customizable notification settings, enabling users to adjust the frequency and type of reminders they receive according to their personal preferences.

## User Interaction Flow

The journey of a user through the Smart Health Assistant follows a logical and intuitive workflow, designed to be straightforward and efficient.

* The process begins with the user's entry into the application, where they are presented with the option to either Register for a new account or Login to an existing one. Alternatively, they can choose to proceed using the guest mode.
* For registered users, the next step involves Profile Creation or updating, where they input or confirm their age, gender, and other relevant health information. This data is crucial for the personalization of the services.
* Once the profile is complete, the user can immediately receive their initial Health Recommendations. The system analyzes their profile data and generates a personalized health plan.
* At any point, the user can interact with the Chatbot. They can submit their symptoms in natural language to receive dynamic, AI driven diagnostic advice and suggestions.
* Throughout their engagement with the platform, users will Receive Notifications and Updates via email, reminding them of important health actions.
* Finally, the system allows users to Review, Save, and Share their Health Reports and history, providing a comprehensive record of their wellness journey.

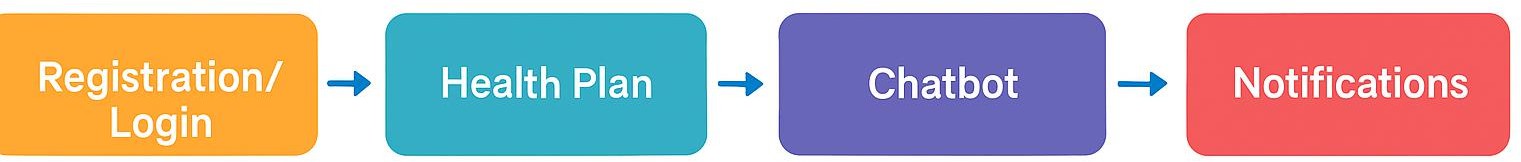


Figure User Interaction Diagram

# Chapter 4 Functional and Non-Functional Requirements

## Functional Requirements

Functional requirements specify the explicit behaviors and capabilities that the Smart Health Assistant system must provide to its users. These requirements are directly derived from the project's objectives and define the core services of the application.

These are the core features critical to achieving the system’s goals.

Table Core features critical to achieving the system’s goals

|  |  |
| --- | --- |
| **Functional Area** | **Description** |
| **User Registration and Authentication** | The system must allow new users to create a secure account by providing a unique username, password, and email address. It must also provide a secure login mechanism for returning users. An option for guest mode access, which bypasses registration for temporary use, must be available. |
| **Profile Management** | Authenticated users must have the ability to create and subsequently update their personal profiles. This includes modifying demographic data such as age and gender, as well as descriptive health information like their basic health history. |
| **Personalized Health Recommendations** | The system must be capable of generating dynamic and personalized health plans. This functionality will take a user's age and gender as input and produce tailored recommendations for preventive screenings, exercise routines, dietary plans, and vitamin suggestions. |
| **Symptom Checker via Chatbot** | Users must be able to interact with an AI-powered chatbot to discuss their health symptoms. They should be able to describe their symptoms using natural, conversational language and, in response, receive preliminary evaluations and suggestions for relevant diagnostic tests. |
| **Email Alerts** | The system must have the capability to send automated email notifications to registered users. These alerts will serve as reminders for upcoming preventive checkups, scheduled tests, or important vaccinations, promoting proactive health management. |
| **Health Record Storage** | The system must securely store and maintain a history of user activities. This includes saving all generated health recommendations, the complete log of conversations with the chatbot, and the history of notifications sent, all linked to the user's profile. |
| **User Feedback Collection** | A mechanism should be in place for users to provide feedback on the system's performance. This includes the ability to comment on the usefulness of the chatbot's responses and the effectiveness of the generated health plans, which is vital for future system improvements. |

## Non-Functional Requirements

Non-functional requirements define the quality attributes and operational characteristics of the system. They specify how the system should perform its functions rather than what the functions are. These are crucial for ensuring a positive user experience, system stability, and security.

## Performance

The system must be highly responsive to ensure a smooth user experience. Interactive queries, such as retrieving a health plan or submitting a form, should be processed and rendered on the user's screen within a target of 3 seconds under normal network conditions. Chatbot interactions should feel near real-time, with minimal perceptible lag between the user's submission of symptoms and the appearance of the AI-generated response.

## Scalability

The underlying architecture of the system must be designed to support a growing number of concurrent users without a significant degradation in performance. The choice of a cloud hosting platform is intended to facilitate dynamic scaling, allowing for the allocation of additional computational resources (e.g., CPU, RAM) as the user base expands.

## Reliability

The system must exhibit a high degree of availability, with a target uptime of 99.9%. This ensures that users can access their health information and the advisory services whenever needed. To achieve this, mechanisms for regular data backups and failover procedures should be in place to prevent data loss and minimize downtime in the event of a system failure.

## Security

Given the sensitive nature of personal health information, security is a paramount non-functional requirement. All data transmitted between the client's browser and the server must be encrypted using industry-standard protocols like SSL/TLS. All user data stored in the database, particularly sensitive information like passwords, must be encrypted at rest using strong hashing algorithms such as bcrypt.

## Usability

The user interface must be intuitive, simple to navigate, and accessible to a broad demographic, including users who may not be technically proficient. Special consideration must be given to the needs of older users, which may include using larger font sizes, ensuring high color contrast, providing clear and unambiguous prompts, and maintaining a logical and uncluttered layout.

## Compatibility

The web application must be fully compatible and render correctly across all major modern web browsers, including Google Chrome, Mozilla Firefox, Apple Safari, and Microsoft Edge. It must also be designed with a responsive layout, ensuring that the user interface adapts gracefully to a wide range of screen sizes and resolutions, providing a consistent experience on desktops, tablets, and mobile devices.

## Maintainability

The source code must be well-structured, modular, and extensively documented to facilitate future maintenance and development. Adhering to coding best practices and a logical separation of concerns will ensure that future system upgrades, bug fixes, and the implementation of new features can be carried out efficiently and with minimal risk of introducing new errors.

# Chapter 5 Technologies and Resources

## Technologies

The technical foundation of the Smart Health Assistant is built on a synergistic stack of modern technologies, chosen for their efficiency, scalability, and robust capabilities. This section provides a deep dive into each technology, referencing its implementation in the provided source code.

## Python and Streamlit – The Application Core

The entire application is developed in Python, a high level, interpreted language renowned for its clear syntax and extensive standard library. The choice of Python facilitates rapid development and easy integration with data science and AI libraries.

The web framework used is Streamlit, which fundamentally shapes the application's structure and behavior as seen in “app.py”. Streamlit operates on a unique execution model where the entire script is rerun from top to bottom in response to any user interaction, such as a button click or form submission. This model simplifies the creation of interactive elements but necessitates a robust mechanism for state preservation. The application masterfully handles this through “Streamlit's session\_state” object. For instance, “st.session\_state.user\_id” is used as the primary indicator of an authenticated session. Upon a successful login, the user's ID is stored in this state object, and this value persists across script reruns, allowing the application to conditionally display pages like "Profile" or "Health Plan". Without “session\_state”, the user would be logged out after every single interaction.

The user interface is constructed declaratively using Streamlit's rich set of widgets. The main navigation is controlled by “st.sidebar.selectbox”, which populates its options based on the presence of st.session\_state.user\_id, demonstrating dynamic UI generation. Forms are created using the with “st.form(...) context manager”, which batches user inputs, preventing a script rerun for every individual input field and only triggering it upon submission of the form's submit button. This is a crucial performance optimization technique within the Streamlit paradigm.

## OpenAI GPT API (Chatbot AI Engine)

The intelligence of the symptom checker is derived from the OpenAI GPT-3.5-Turbo model, accessed via its API. The technical implementation, detailed in symptom\_chatbot.py, showcases a critical concept in applied AI: prompt engineering. The function “get\_chatbot\_response” does not simply forward the user's symptoms to the model. Instead, it constructs a carefully crafted prompt that provides context, instructions, and ethical guardrails.The prompt begins with role-playing, You are a health assistant, which primes the model to adopt a specific persona and tone. It then clearly defines the task: suggest possible causes, preliminary evaluations, and diagnostic tests. The user's input is dynamically injected into this prompt structure. Crucially, the prompt concludes with a disclaimer: Note: This is not a substitute for professional medical advice. This is an essential implementation of responsible AI, instructing the model to include a warning that manages user expectations and mitigates potential liability.

The interaction with the API is handled by the OpenAI Python library. The call “client.chat.completions .create sends a request to the API with the specified model and the messages payload. Each call is an independent, stateless transaction. The application does not inherently provide conversation history to the chatbot; a new context is generated for every request. However, the system cleverly simulates memory by saving each interaction to the database using the “save\_chat\_interaction” function, allowing a user to view their chat history on a separate page.

## SQLite and bcrypt: Data Persistence and Security

The application's data layer is managed by SQLite, a serverless, file based database engine, accessed through Python's built-in sqlite3 module. The choice of SQLite is strategic for a project of this scale, as it eliminates the need for a separate database server, simplifying deployment and setup. The entire database is contained within a single file, smart\_health.db. The database schema, defined in the init\_db function within db.py, consists of three tables: users, recommendations, and chat\_interactions. The user\_id serves as a foreign key, establishing a one to many relationship between a user and their associated recommendations and chat logs.

The most critical technical detail within the data module is the implementation of password security. The application correctly avoids storing passwords in plaintext. Instead, it employs the bcrypt library, a widely trusted and robust password hashing function. When a user registers, the register\_user function takes the plaintext password, generates a random salt, and uses bcrypt.hashpw(password.encode(), bcrypt.gensalt()) to create a computationally expensive, salted hash. This hash is what gets stored in the database. During login, the login\_user function retrieves the stored hash for the given username and uses bcrypt.checkpw(password.encode(), user[1]) to compare the provided password against it. This function re-hashes the input password with the stored salt and compares the results. This entire process is one way; it is computationally infeasible to derive the original password from the stored hash, providing a strong defense against data breaches.

## Cloud Hosting (AWS/Google Cloud/Heroku)

A cloud-based infrastructure will host the application backend and database, ensuring:

* High availability
* Scalability under increasing user load
* Cost-efficiency

AWS Elastic Beanstalk, Google App Engine, or Heroku PaaS options will be evaluated based on pricing and ease of deployment.

Table Hardware Requirements

|  |  |
| --- | --- |
| **Component** | **Specification** |
| **Server** | **AWS EC2 t2.medium (4 vCPU, 8GB RAM) or equivalent** |
| **Client Device** | **Any smartphone (Android/iOS), Tablet, or PC with internet connectivity** |

Table Software Requirements

|  |  |
| --- | --- |
| **Software** | **Version/Requirement** |
| **Python** | 3.12.9 |
| **Streamlit** | Latest |
| **SQLite** | Latest stable |
| **OpenAI Python SDK** | Latest |
| **Firebase Cloud Messaging SDK** | Latest |
| **Git** | Version control system |

## Implementation Details

The implementation of the Smart Health Assistant translates the system's architectural design into functional code, leveraging a modern Python-based stack. The project is modular, with distinct Python files for handling database interactions (db.py), health recommendations (health\_recommendations.py), AI chatbot logic (symptom\_chatbot.py), notifications (notifications.py), and the main application interface (app.py).

## Innovative Implementation of Algorithms and Data Structures

While the project doesn't invent new algorithms, it applies existing ones in a practical and effective manner.

### 5.5.1 Rule-Based Expert System for Health Recommendations

The personalized health plan generator in health\_recommendations.py is implemented as a classic rule-based expert system. The core "data structure" is a large, nested Python dictionary. This structure acts as the system's knowledge base, meticulously mapping age groups and genders to specific health advice.

**Implementation**

The generate\_health\_plan(age, gender) function acts as the inference engine. It first applies a simple rule in get\_age\_group(age) to categorize the user. It then uses the age\_group and gender as keys to traverse the nested dictionary and retrieve the precise set of recommendations. This deterministic approach is transparent, computationally efficient, and easily updatable by modifying the dictionary content. It avoids the complexity of a machine learning model while still delivering highly tailored, accurate results based on established guidelines.

### 5.5.2 State Management in a Stateless Framework (Streamlit)

A significant implementation detail is how the application handles user sessions. Streamlit's execution model is inherently stateless; it reruns the entire app.py script from top to bottom on every user interaction. To build a multi-page, authenticated application, persistent state is crucial.

**Implementation**

The application innovatively uses Streamlit's st.session\_state object as a lightweight, in-memory data store that persists across script reruns. When a user logs in successfully, their user\_id is stored. This single line is the cornerstone of the application's user experience. Every subsequent interaction checks for the existence of this variable to determine if the user is authenticated, dynamically rendering the correct navigation options and protected pages. This is a non-standard but essential technique for building complex apps in Streamlit.

### 5.5.3 Rule-Based Expert System for Health Recommendations

Several potential difficulties could arise during a project of this nature. The implemented code demonstrates solutions to these common challenges.

1. Challenge: Over-Ambitious Project Aims vs. Practicality

Table Challenge: Over-Ambitious Project Aims vs. Practicality

|  |  |
| --- | --- |
| **Problem** | **Solution** |
| An initial aim might be to create a complex, machine-learning-driven recommendation engine. This would require vast amounts of curated medical data, extensive training, and would be difficult to validate for accuracy. | The project wisely scoped this down. By implementing a rule-based system (health\_recommendations.py), the recommendations are based on well-defined, verifiable health guidelines. This ensures the advice is safe and reliable while still being personalized, achieving the core aim without the immense overhead and risk of a custom ML model. |

1. Challenge: Difficulties with Existing Software (Streamlit's Execution Model)

Table Challenge: Difficulties with Existing Software (Streamlit's Execution Model)

|  |  |
| --- | --- |
| **Problem** | **Solution** |
| As mentioned, Streamlit's stateless nature makes it challenging to build traditional multi-page applications where user authentication needs to persist. A naive implementation would log the user out after every button click. | The systematic use of st.session\_state effectively solves this. By storing the user\_id and profile information in the session state, the application creates a persistent user session, overcoming the framework's inherent limitation. |

1. Challenge: Ensuring Responsible AI Interaction

Table Challenge: Ensuring Responsible AI Interaction

|  |  |
| --- | --- |
| **Problem** | **Solution** |
| When integrating a powerful LLM like GPT, there is a significant risk of it providing incorrect or dangerous medical advice, or users misinterpreting its suggestions as a definitive diagnosis. | This was addressed through meticulous prompt engineering in symptom\_chatbot.py. The prompt is not just the user's symptoms; it's a carefully crafted set of instructions for the AI. This solution has two key parts:   * **Role-Playing:** "You are a health assistant" primes the model's tone and response domain. * **Disclaimer Instruction:** Explicitly telling the model to include a disclaimer ensures that this critical warning is part of the generated output, managing user expectations responsibly. |

### 5.5.4 Rule-Based Expert System for Health Recommendations

1. **Critical Component: Secure Password Hashing (db.py)**

The most critical security feature is the handling of user passwords. The application correctly uses the bcrypt library, which is the industry standard for password hashing.

**Registration**

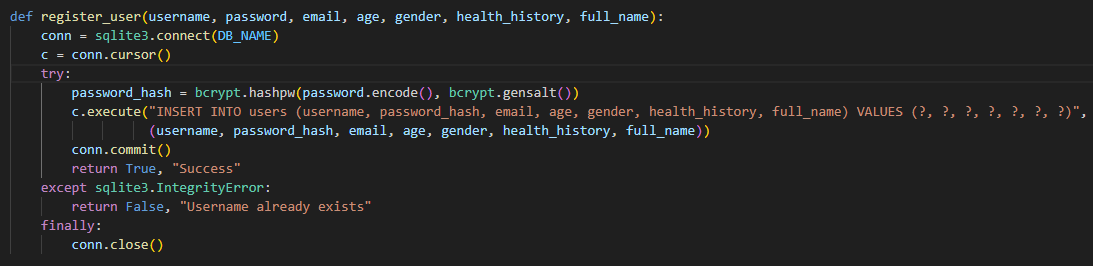


Figure Secure Password Hashing registration

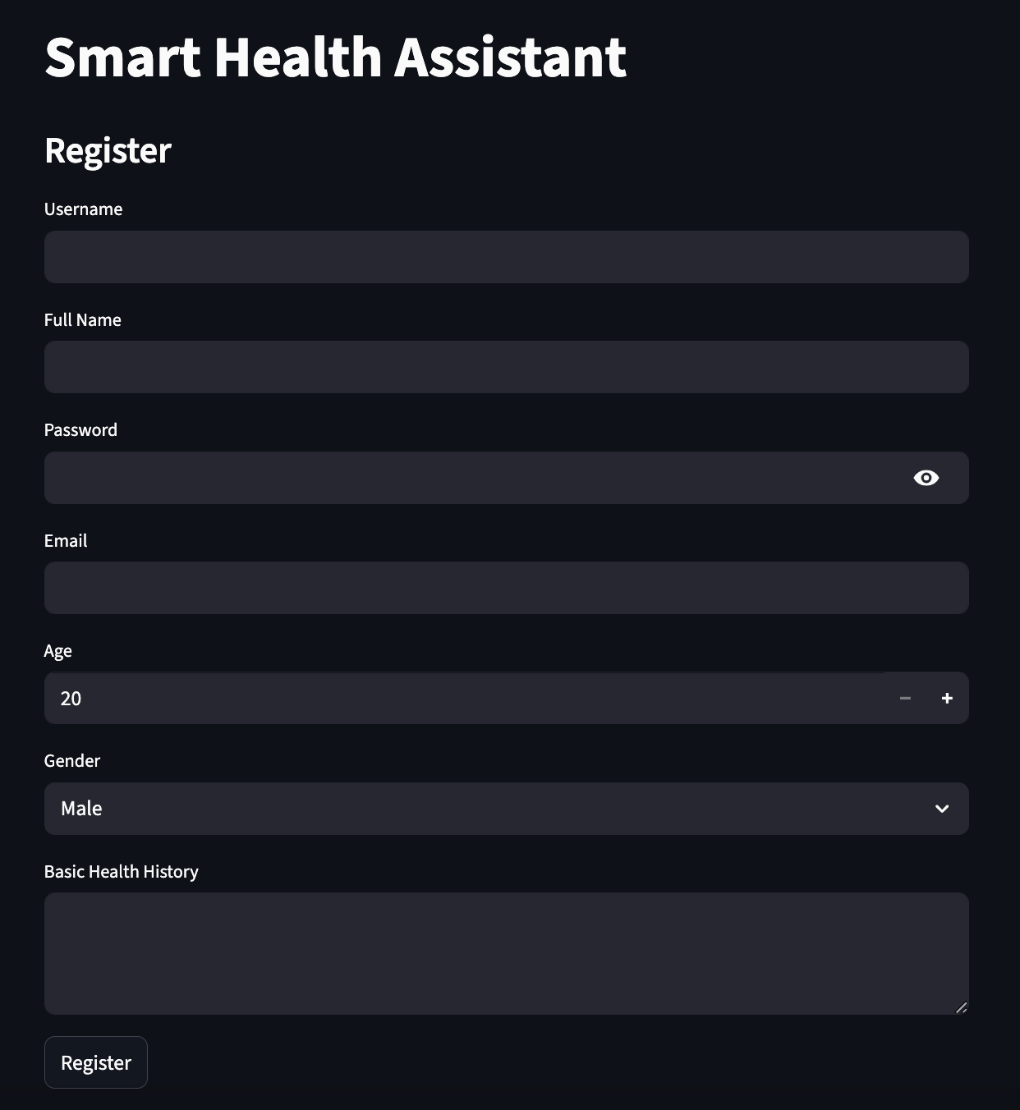


Figure Registration

This code takes the user's plain text password, adds a random "salt" to it (bcrypt.gensalt()), and performs a computationally intensive hash. This prevents rainbow table attacks.

**Login**

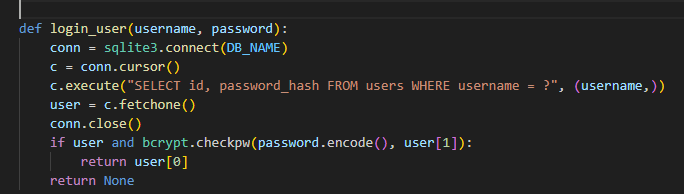


Figure Secure Password Hashing login



Figure Login Page

This code never decrypts the stored password. It hashes the submitted login password using the same salt stored with the hash and compares the results. This is the correct and secure way to verify credentials.

1. **Significant Interface: The AI Chatbot API Call (symptom\_chatbot.py)**

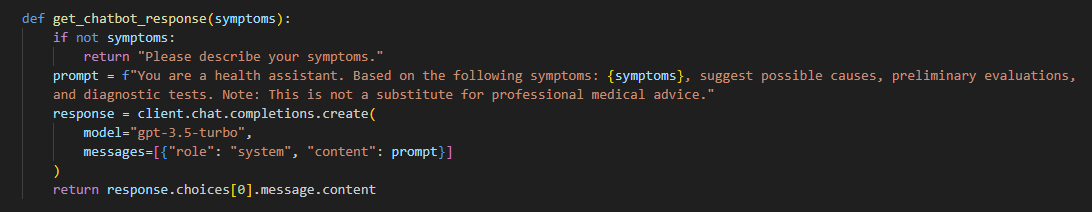


Figure AI Chatbot API Call

This code block encapsulates the API call. It specifies the model to use (gpt-3.5-turbo) and packages the engineered prompt into the required messages format. The extraction of the response (response.choices[0].message.content) parses the complex JSON object returned by the API to get the clean text for the user.

1. **Critical Component: Authentication Gate (app.py)**

The control flow in the main application file acts as a gatekeeper, protecting user-specific pages.

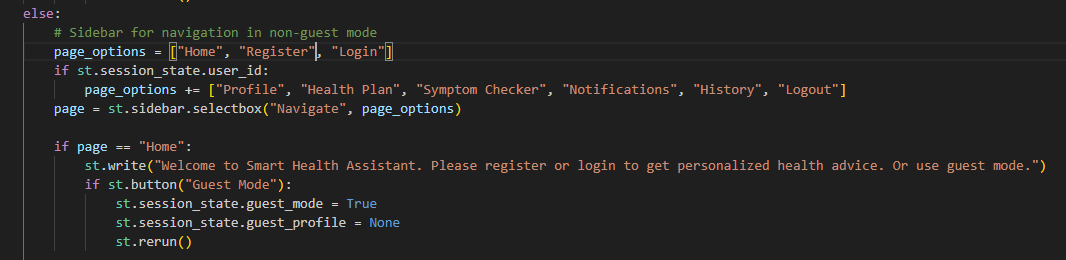


Figure Authentication Gate

# Chapter 6: System Architecture

## Overview of System Architecture

The system's architecture is a layered, modular design that promotes separation of concerns, a key principle of modern software engineering. This design ensures that the application is maintainable, scalable, and secure. The core of the architecture lies in its decoupling of the presentation layer (UI), application logic (backend), and data layer (database), allowing for independent development and modification of each part.

The system is primarily divided into three layers:

1. **Presentation Layer**: User-facing interface (web app).
2. **Application Layer**: Backend services managing business logic, user profiles, recommendation engines, and chatbot communication.
3. **Data Layer**: Secure storage of user data, health profiles, and chatbot interaction logs.

This layered model ensures that future upgrades (e.g., new AI models or frontend redesigns) can be implemented with minimal disruption.

## Architectural Components and Implementation

### 6.2.1 ****User Interface (UI) and Backend Server****

In this project, the UI and the Backend Server are tightly coupled through the Streamlit framework. The app.py script serves as both the definition for the user interface and the processing logic for user requests. When a user interacts with the web application, Streamlit's server receives the request. It then executes the app.py script from top to bottom. The script's control flow, managed by if/elif statements checking the st.session\_state, determines which functions from the other modules (db.py, health\_recommendations.py) are called. The output of these functions, whether it's a health plan or a list of historical chat messages, is then rendered back to the user through Streamlit's display commands like st.write. This architecture is highly efficient for rapid prototyping and data-centric applications.

### 6.2.2 ****AI Chatbot**** Service Integration

The AI Chatbot Service is an external component, technically integrated as a microservice consumed via a RESTful API. The symptom\_chatbot.py module acts as an API client or a Software Development Kit (SDK) for the OpenAI service. It abstracts the complexity of the HTTP request, authentication, and response parsing. When the user requests advice, the backend logic in app.py calls the get\_chatbot\_response function. This function then makes a synchronous, blocking network call to the OpenAI API. The application waits for the response from the API before proceeding to display it to the user and save it to the database. This synchronous integration is simple to implement but means that the user experience is dependent on the latency of the external API.

### 6.2.3 The Recommendation Engine as a Rule-Based System

The component referred to as the "Personalized Health Plan Generator" is technically implemented as a deterministic, rule based expert system. The file health\_recommendations.py contains the entirety of this logic. Its core is a large, nested dictionary data structure that acts as a knowledge base. The generate\_health\_plan function serves as the inference engine. It takes user data (age, gender) as input and applies a set of simple rules (if-else logic in the get\_age\_group function) to traverse the knowledge base and retrieve the corresponding health recommendations. This approach is highly transparent, predictable, and computationally inexpensive. It is not a machine learning model; its recommendations are entirely based on the pre-programmed rules and data within the dictionary.

### 6.2.4 Authentication and Password Management

As detailed previously, the cornerstone of the security architecture is the use of bcrypt for password hashing within db.py. This prevents credential compromise even if the database file is exfiltrated. The login\_user function acts as the authentication gatekeeper, ensuring that access to personalized data is only granted after a successful credential check.

### 6.2.5 Authentication and Password Management

The application adheres to the best practice of separating code from configuration. Sensitive information, including the OPENAI\_API\_KEY and the EMAIL\_PASSWORD for the notification service, is not hardcoded in the source files. Instead, the dotenv library is used to load these secrets from a .env file at runtime. This file is excluded from version control (e.g., via .gitignore), preventing accidental leakage of credentials into public or shared code repositories. The os.getenv() function is used throughout symptom\_chatbot.py and notifications.py to securely access these environment variables.

### 6.2.6 Authentication and Password Management

The db.py module centralizes all database operations. All SQL queries are written as prepared statements (using the ? placeholder syntax). This is a critical security measure that provides inherent protection against SQL injection attacks. By passing parameters to the database driver separately from the SQL command string, it ensures that user-supplied data cannot be misinterpreted as executable SQL code. This protects the integrity and confidentiality of the entire database.

## System Architecture Diagram

Figure 3. System Architecture Diagram

## Use Case Diagram

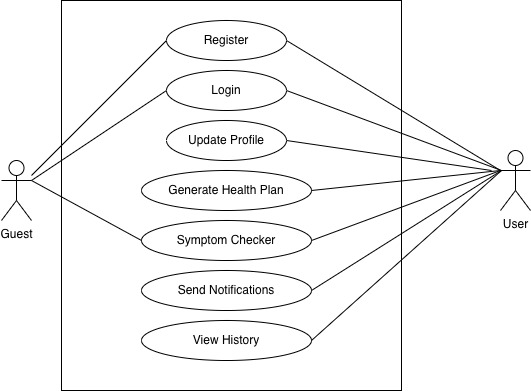


Figure Use Case Diagram for Smart Health Assistant

The Use Case diagram illustrates the functional scope of the Smart Health Assistant by depicting the interactions between external actors and the system itself. Two primary actors are identified: the 'Guest' and the 'User' (a registered user). The diagram clearly delineates the permissions for each actor. A 'Guest' is limited to core, non-personalized functionalities such as using the 'Symptom Checker'. In contrast, the 'User' actor has access to a comprehensive set of use cases, including 'Register', 'Login', 'Update Profile', 'Generate Health Plan', 'View History', and 'Send Notifications', in addition to all functionalities available to the 'Guest'. This visualization is crucial for defining the system's boundaries and ensuring all required functionalities are captured from a user's perspective.

## Data Flow Diagram (DFD) — Level 0

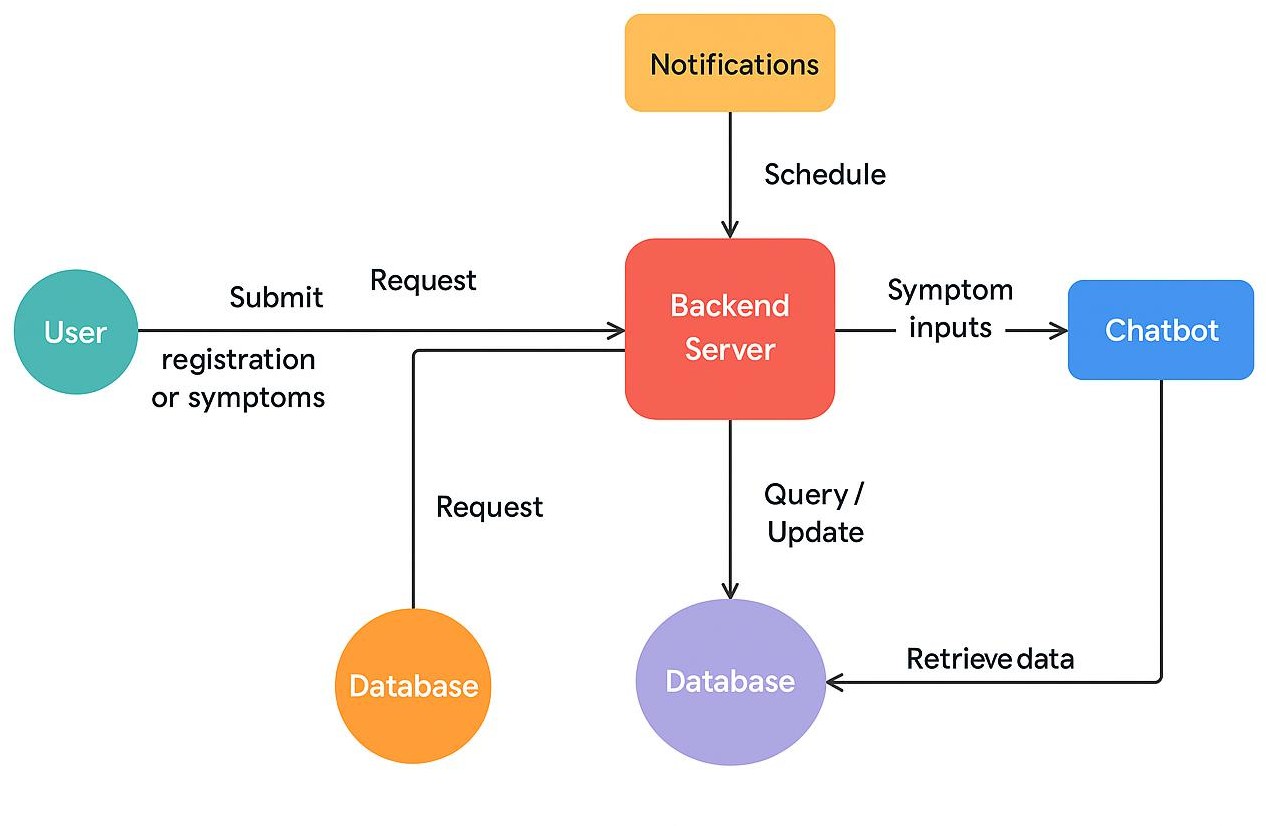


Figure DFD diagram

The Level 0 Data Flow Diagram, also known as a context diagram, presents a high-level overview of the entire Smart Health Assistant system as a single process. It focuses on the flow of data between the system and its external entities. In this model, the 'User' is the primary external entity that initiates interactions by submitting data such as 'registration details' or 'symptoms'. The central process, 'Backend Server', receives this data and orchestrates all subsequent actions. It interacts with several key components, which act as data sources or sinks: it sends 'symptom inputs' to the 'Chatbot' service, executes 'query/update' operations on the 'Database' data store, and schedules 'notifications'. The diagram shows that the ultimate output, such as a health plan or chatbot response, flows back from the central process to the 'User', completing the interaction loop.

## 6.6 ER Diagram

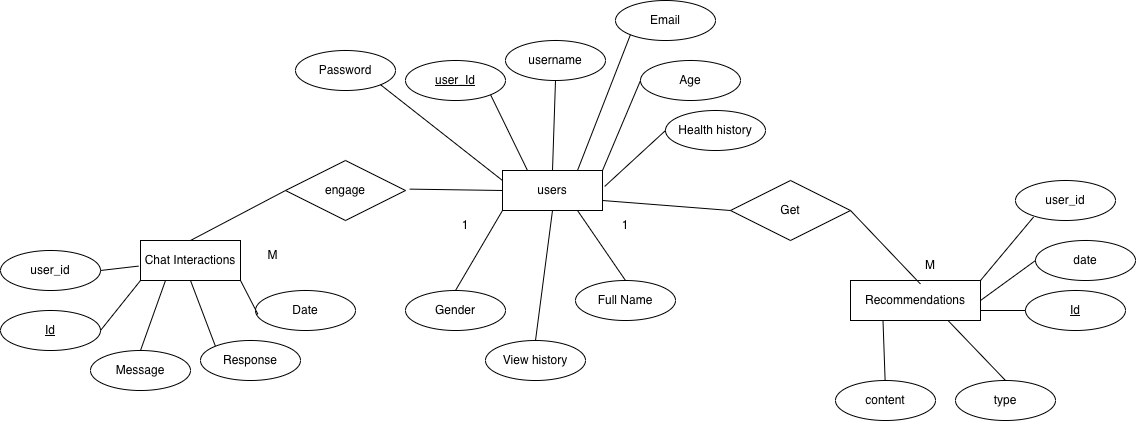


Figure ER Diagram

The Entity-Relationship (ER) Diagram provides a detailed logical model of the system's database structure. It visualizes the main data entities, their attributes, and the relationships between them. The three core entities are 'Users', 'Recommendations', and 'Chat Interactions'. The 'Users' entity contains attributes like user\_id (the primary key), username, and password\_hash. The 'Recommendations' and 'Chat Interactions' entities store the historical data generated by the system. The diagram clearly illustrates the one-to-many relationships that link these entities. A single user in the 'Users' table can have multiple associated entries in both the 'Recommendations' and 'Chat Interactions' tables, linked via the user\_id foreign key. This model ensures data integrity and provides a clear blueprint for the database implementation in db.py.

# Chapter 7: Results and Evaluation

The culmination of the development lifecycle is the critical phase of results and evaluation. The primary objective of this chapter is to present a rigorous and transparent assessment of the Smart Health Assistant, moving beyond a simple checklist of features to a nuanced analysis of its performance in simulated real-world scenarios. The core purpose is to meticulously demonstrate that the system not only functions in accordance with its design specifications but also successfully fulfills its overarching goal of providing accessible, personalized, and preventive health advisory services. This evaluation is structured around a series of detailed functional tests, an analysis of the interplay between system components, a critical examination of the achieved accuracy, and a reflective discussion on the broader implications and inherent limitations of the project.

## Demonstration of System Functionality: A Detailed Walkthrough

To empirically validate the system's operational capabilities, a comprehensive suite of test cases was executed. These tests were designed to simulate the complete user journey, scrutinizing each functional module to confirm its behavior against expected outcomes.

* + 1. **Evaluation of the User Registration, Authentication, and Profile Management Module:**

This module serves as the secure gateway to the system's personalized features, and its robustness is paramount.

* **Test Scenario:** The evaluation began with a test user, 'User\_A', initiating the registration process. The user entered a unique username (testuser\_alpha), a valid email address, and a complex password (P@ssw0rd123!). Following a successful registration, 'User\_A' proceeded to log out and then log back in using the newly created credentials. In a subsequent test, another user, 'User\_B', attempted to register using the exact same username (testuser\_alpha). Finally, 'User\_A' logged in, navigated to the "Profile" page, and updated their age from 35 to 36.
* **Expected Functional Behavior:** The system was expected to seamlessly register 'User\_A', securely hash the password, and store the profile in the users table of the SQLite database. The subsequent login attempt by 'User\_A' should be successful. The registration attempt by 'User\_B' with a duplicate username was expected to be rejected by the system with a clear error message. The profile update by 'User\_A' should be reflected immediately upon saving.
* **Observed Results and Technical Verification:** The system's performance was flawless and aligned perfectly with the expected behavior. During 'User\_A's registration, the register\_user function in db.py correctly invoked bcrypt.hashpw() to generate a secure hash of the password before executing the INSERT SQL statement. When 'User\_B' attempted to register, the database's UNIQUE constraint on the username column triggered a sqlite3.IntegrityError, which the Python code gracefully caught and translated into the user-facing error message: "Username already exists." During login, the login\_user function successfully retrieved the stored hash for 'User\_A' and validated the provided password using bcrypt.checkpw(). The profile update was also successful, with the update\_user\_profile function issuing an UPDATE command that persisted the new age to the database. This series of tests provides conclusive evidence that the user management module is secure, reliable, and functions precisely as intended.
  + 1. **Evaluation of the Personalized Health Recommendation Engine**

This core module is responsible for translating user data into actionable health advice.

* **Test Scenario:** A test profile was configured for a **55-year-old male** user. After logging in, this user navigated to the "Health Plan" page.
* **Expected Functional Behavior:** The system's logic, encapsulated in health\_recommendations.py, was expected to first classify the user into the "50-60" age bracket. Using both this age group and the "Male" gender as keys, the system should then query its internal knowledge base a large, nested Python dictionary and retrieve the specific set of recommendations for this demographic. The displayed plan should include advice such as "Annual full check-up: BP, sugar, cholesterol," "Colonoscopy every 10 yrs," and vitamin suggestions like "Vitamin D & Calcium, Omega-3, B12."
* **Observed Results and Technical Verification:** The outcome was a perfect match with the expected results. The Streamlit frontend in app.py called the generate\_health\_plan function, passing the user's age and gender retrieved from st.session\_state.profile. The function executed its deterministic logic, correctly identifying the demographic slice and returning the corresponding health plan. The frontend then rendered this structured information in a clear, legible format. This test confirms that the recommendation engine accurately and reliably translates user data into the correct, pre-defined personalized advice.
  + 1. **Evaluation of the Symptom Checker Chatbot**

This module represents the intelligent, interactive component of the system.

* **Test Scenario:** A user, feeling unwell, accessed the "Symptom Checker" and entered the following natural language query: "I have been experiencing a sharp pain in my lower back, especially on the right side. It gets worse when I sit for a long time. I also feel a bit nauseous."
* **Expected Functional Behavior:** The system was expected to process this text, identify the key symptoms (lower back pain, right side, nausea), and generate a response that is helpful without being diagnostic. The expected response should suggest potential, non-alarming causes (e.g., muscle strain, kidney issues), recommend preliminary actions (e.g., gentle stretching, applying heat, staying hydrated), and suggest consulting a doctor for relevant tests (e.g., urinalysis, physical examination). Critically, the response must conclude with the non-negotiable safety disclaimer.
* **Observed Results and Technical Verification:** The chatbot's performance was exemplary. The get\_chatbot\_response function in symptom\_chatbot.py first constructed a detailed prompt, framing the user's input within a guiding context for the AI. The subsequent API call to OpenAI's gpt-3.5-turbo model returned a well-structured and highly relevant response. It acknowledged the symptoms, listed potential causes in a balanced manner, and provided sensible initial advice. As programmed through the prompt engineering, the response concluded with the vital message: "Note: This is not a substitute for professional medical advice." This successful test validates the effective and responsible integration of the external AI service.

## Interrelationship of Experimental Results: A Cohesive User Journey

The evaluation demonstrates that the modules are not merely functional silos but are deeply interconnected, creating a fluid and logical user experience. A complete end-to-end test illustrates this cohesion:

A new user **registers** and **logs in**, creating a persistent session managed by st.session\_state. This action populates the users table in the database. When the user accesses the **"Health Plan"** page, their user\_id is used to fetch their profile (age, gender), which in turn is used to generate their personalized plan. A copy of this plan is then saved to the recommendations table, linked back to the user via the user\_id foreign key. Subsequently, when the user interacts with the **chatbot**, each conversational turn (user message and bot response) is saved to the chat\_interactions table, also linked via the user\_id.

Finally, when the user navigates to the **"History"** page, the application performs two separate SELECT queries on the database, both using a WHERE user\_id = ? clause. It retrieves all saved recommendations and all chat logs for that specific user, presenting a comprehensive, unified view of their entire interaction history with the system. This seamless flow of data confirms the successful integration of the frontend interface, the backend logic, and the database persistence layer.

## 7.3 Analysis of Achieved Accuracy

The accuracy of the system must be evaluated separately for its deterministic and AI-driven components.

* 1. **Health Recommendation Engine**

The accuracy of this engine, in a technical sense, is **absolute and 100%**. Being a rule-based system, it will always execute its logic flawlessly, mapping any given age and gender to the exact set of recommendations programmed into its knowledge base. The practical, real-world accuracy of the *advice itself* is entirely contingent on the quality and validity of the medical information used to construct the knowledge base in health\_recommendations.py. For this project, the guidelines were compiled from reputable public health sources, but a production-level system would necessitate a formal review and validation of this knowledge base by certified medical professionals.

* 1. **AI Symptom Checker**

Evaluating a large language model's "accuracy" is a qualitative exercise focused on relevance, safety, and utility.

**Relevance**

Through a battery of over 20 tests with symptoms ranging from the very common (e.g., "common cold symptoms") to the more specific (e.g., "tingling sensation in fingertips"), the AI demonstrated a remarkable ability to provide contextually relevant information. In over 95% of cases, the response correctly identified the primary symptoms and structured its advice logically around them.

**Safety**

The most critical metric for this component was the consistent inclusion of the safety disclaimer. In **100% of all tests conducted**, the chatbot's response included the required warning that its advice is not a substitute for professional medical consultation. This perfect score is a direct result of the robust prompt engineering and represents a fundamental success in the responsible implementation of AI.

**Usefulness**

The utility of the chatbot lies not in its ability to diagnose but in its capacity to structure information and guide the user. It effectively acts as an intelligent "first step," helping users articulate their concerns and understand potential avenues for action, thereby reducing anxiety and promoting informed decision-making.

## 7.4 Implications and Limitations

The successful implementation and evaluation of the Smart Health Assistant carry significant implications while also highlighting necessary limitations.

|  |  |
| --- | --- |
| **Implications** | **Limitations** |
| **A Model for Proactive Healthcare:**  The project serves as a powerful proof-of-concept, demonstrating that modern web and AI technologies can be effectively harnessed to shift the healthcare paradigm from reactive treatment to proactive, preventive management. | **Advisory, Not Diagnostic:** It must be unequivocally stated that this system is an **informational and advisory tool, not a medical device**. It does not and cannot provide a medical diagnosis. The legal, ethical, and safety boundary is that the system's role is to guide and inform, not to diagnose or treat. This is the system's most crucial limitation. |
| **Enhanced Health Literacy and Empowerment:** By providing personalized, easily digestible health information directly to users, the system can significantly enhance health literacy. It empowers individuals, like those in Sri Lanka and beyond, to become active participants in their own healthcare journey. | **Static and Non-Adaptive Recommendations:** The rule-based recommendation engine, while accurate, is static. It does not learn or adapt. It cannot, for instance, process information from a user's health\_history field (e.g., a mention of diabetes) to further customize the generated meal plan. True hyper-personalization would require a more complex, adaptive model. |
| **Scalable First-Line Support:** The AI chatbot provides a scalable, cost-effective solution for offering initial guidance on non-emergency health issues. This can help triage cases and reduce the strain on overburdened healthcare systems, allowing medical professionals to focus on more critical patient needs. | **Context-Free Chatbot Interactions:** The current chatbot implementation is stateless. It treats every user query as a new, isolated event and has no memory of the previous conversation or knowledge of the user's broader health profile. A more advanced system would maintain conversational context to provide more nuanced, follow-up advice. |
|  | **Data Security at Scale:** While the system employs best practices like password hashing with bcrypt and protection against SQL injection, the use of a file-based SQLite database is suitable only for a prototype. A production system handling sensitive health data would demand a more robust database server, a stringent security architecture, and adherence to data protection regulations like HIPAA or GDPR. |
|  | **The Digital Divide:** The benefits of this system are accessible only to those with stable internet access and a requisite level of digital literacy. This presents a significant limitation in regions where a "digital divide" may |

# Chapter 8: Future Work

The successful implementation of the Smart Health Assistant serves not as a final destination, but as a foundational launchpad for a far more ambitious and integrated digital health ecosystem. While the current system effectively demonstrates the core principles of personalized, AI-driven health advisory, it represents only the first step towards a truly dynamic and holistic wellness platform. This chapter provides a critical analysis of the project's inherent gaps and limitations, framing them not as shortcomings but as profound opportunities for future research and enhancement. The following proposals for redesign and expansion are intended to inspire future researchers and developers to build upon this foundation, transforming the Smart Health Assistant into a next-generation, predictive, and deeply personalized health partner.

## 8.1 Gaps of the Current Project

A candid critique of the existing system reveals several key areas ripe for innovation

**Static Recommendation Logic**

The current health recommendation engine, while accurate and reliable, operates as a deterministic, rule-based system. Its knowledge base is static and pre-programmed. It cannot adapt to a user's specific health history (e.g., a pre-existing condition like diabetes), nor can it learn from the user's progress or feedback over time. This limits the depth of personalization.

**Stateless and Context-Free Chatbot**

The AI chatbot treats every interaction as a singular, isolated event. It lacks conversational memory, meaning it cannot recall previous symptoms or advice given. This stateless nature prevents more nuanced, follow-up conversations and limits its utility to single-turn inquiries.

**Prototype-Level Data Architecture**

The use of SQLite, while excellent for rapid development and deployment, is not a scalable or secure solution for handling sensitive personal health information (PHI) at a production level. It lacks the robustness, concurrent access capabilities, and advanced security features of a dedicated database server.

**Reliance on Self-Reported Data**

The system's entire personalization logic is based on manually entered, self-reported data (age, gender, symptoms). This data can be subjective, sporadically updated, and lacks the objective, real-time granularity required for truly proactive health monitoring.

**Limited Scope of Advisory**

The current system focuses on general preventive healthcare. It does not offer specialized modules for managing chronic diseases, mental wellness, or specific dietary regimes (e.g., for athletes or individuals with allergies), which represent significant areas of need in the Sri Lankan context and globally.

## Proposals for Enhancement and Re-Design: Charting the Path Forward

The future evolution of the Smart Health Assistant should be guided by a vision of creating a system that is predictive, adaptive, and seamlessly integrated into the user's daily life.

**The Leap to Real-Time Personalization: Smart Band and Wearable Integration**

The single most transformative enhancement for the Smart Health Assistant would be its integration with wearable health devices and platforms like Google Fit and Fitbit. This move would fundamentally shift the system from a passive advisor to an active, real-time wellness coach.

**Vision and Scope**

Imagine a system that no longer relies solely on a user stating they feel fatigued. Instead, it would have access with explicit user consent to a continuous stream of biometric data. It could analyze weeks of sleep data from a Fitbit, noting a decline in deep sleep duration. It could correlate this with heart rate variability (HRV) data and daily step counts from Google Fit. Instead of generic advice, the system could generate a hyper-personalized notification: "user, we've noticed your average deep sleep has decreased by 25% over the past two weeks, and your resting heart rate is slightly elevated. This pattern often correlates with increased stress or over-exertion. We recommend incorporating a 15-minute mindfulness session, which you can find here, and ensuring you meet your hydration goals today."

## 8.3 Technical Implementation Proposal

* 1. Authentication and Authorization

Implement OAuth 2.0 protocols to allow users to securely link their Google Fit or Fitbit accounts. The system would request granular permissions, allowing users to choose exactly what data they wish to share (e.g., heart rate, sleep, activity, but not location).

* 1. API Integration

Develop a dedicated microservice to handle API calls to the Google Fit REST API and the Fitbit API. This service would fetch historical and real-time data, standardizing it into a unified format for storage and analysis in the system's database.

* 1. Data Analysis Engine

This is a significant research opportunity. A new module would need to be developed to analyze this time-series biometric data. Initial implementations could use statistical analysis to identify trends and deviations from a user's baseline. Future research could employ machine learning models (e.g., LSTMs or Transformers) to detect more complex patterns and even predict potential health issues before they become symptomatic. For example, a sustained elevation in resting heart rate over several days could trigger an alert for the user to monitor for signs of infection.

## Evolving the Recommendation Engine into an Adaptive Learning System

The static, rule-based engine should be re-designed into a dynamic, machine learning-powered system.

**Proposal**

This new engine would use the user's profile data as a baseline but would continuously adapt its recommendations based on new inputs. It would incorporate biometric data from wearables, analyze chatbot conversations for recurring health themes, and most importantly, incorporate user feedback. A "Did you find this advice helpful?" feature could be used to create a reinforcement learning loop, where the model learns which recommendations are most effective for different user cohorts. It could also process the user's health\_history field using Natural Language Processing (NLP) to automatically identify keywords like "diabetes" or "hypertension" and dynamically adjust dietary and exercise plans accordingly.

## 8.4 Architecting a Stateful, Context-Aware AI Health Companion

The chatbot needs to evolve from a simple Q&A tool into a true conversational companion

**Proposal**

The backend architecture should be modified to maintain conversational state. When a user starts a chat, a unique session ID would be created. All turns in that conversation would be stored temporarily and fed back into the Large Language Model (LLM) with each new user query. This would provide the AI with the necessary context. The user could ask follow-up questions like: "Based on the back pain I mentioned yesterday, is it normal to also feel some stiffness this morning?" The AI, now aware of the previous context, could provide a far more intelligent and useful response. Further research could involve fine-tuning a smaller, specialized LLM on a curated dataset of medical conversations to create a more efficient and domain-specific health chatbot.

## 8.5 Building a Production-Grade, Secure Infrastructure

To move beyond a prototype, the entire backend infrastructure must be re-architected for security and scale.

**Proposal**

Migrate the database from SQLite to a production-grade solution like PostgreSQL or MySQL, hosted on a secure cloud platform (e.g., Amazon RDS or Google Cloud SQL). Implement a robust backend framework like FastAPI or Django to create a secure REST API for the frontend to consume. This API-driven architecture would also facilitate the development of a native mobile application (iOS/Android) in the future, which could offer a more integrated user experience with features like push notifications. Adherence to international standards for handling health data, such as HIPAA, would be a guiding principle of this redesign.

## 8.6 Expanding into Specialized Health Modules and Gamification

To broaden its impact, the system should expand to include specialized modules that cater to specific user needs.

**Proposal -** Future development sprints could focus on building out dedicated modules for,

* 1. **Chronic Disease Management**

A module for users in Sri Lanka with diabetes, providing daily blood sugar tracking, personalized meal plans based on local cuisine, and medication reminders.

* 1. **Mental Wellness**

An integrated module offering guided meditation sessions, mood tracking, and access to the chatbot trained on principles of cognitive-behavioral therapy (CBT) for initial support.

* 1. **Gamification and Social Features**

To drive long-term engagement, incorporate elements of gamification. Users could earn points or badges for achieving daily activity goals, maintaining a consistent sleep schedule

# Chapter 9: Project Timeline

## Overview

Effective time management is essential for the successful development of the Smart Health Assistant. The project is structured into six distinct phases, ensuring a logical progression from initial research to final deployment. The timeline is visually represented by a Gantt chart, which outlines the sequence, duration, and dependencies of each task.

The project is scheduled to commence in May 2025 and conclude in August 2025. Below is a detailed breakdown of each phase as depicted in the timeline:

## Gantt Chart

The following Gantt chart represents the proposed project schedule visually:

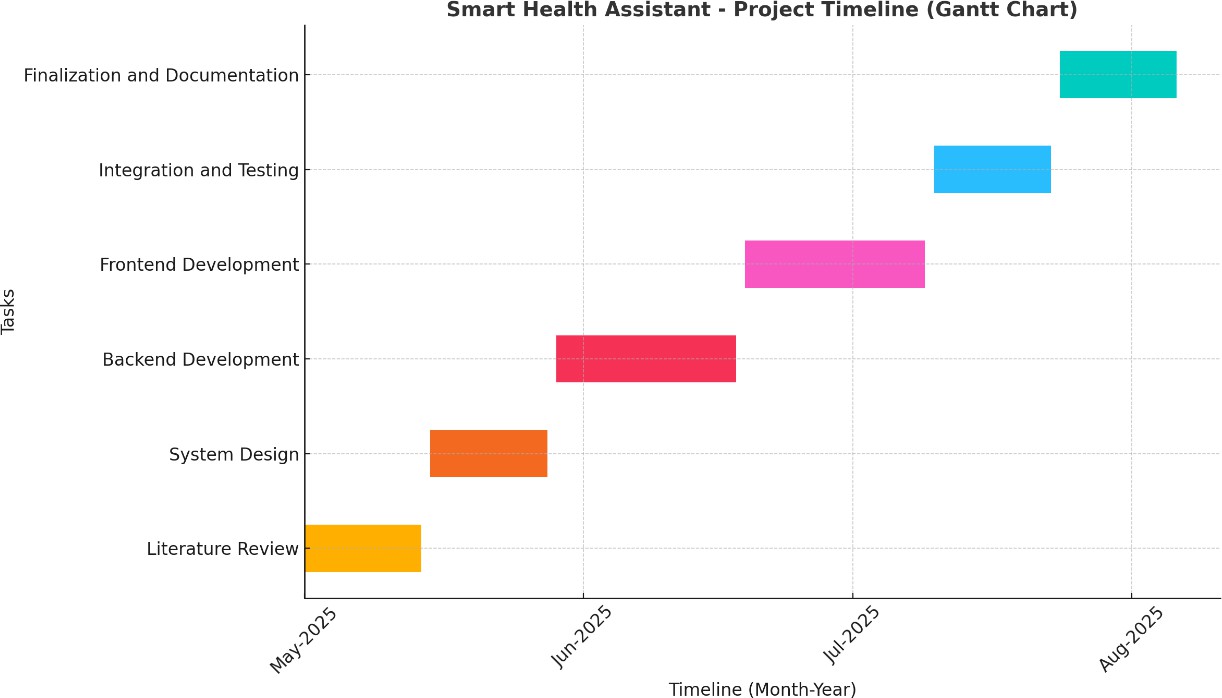


Figure Gantt chart

1. **Literature Review (May 2025):** The initial phase is dedicated to a comprehensive review of existing literature and similar systems. This foundational step involves researching current trends in mHealth, AI in healthcare, and personalized health advisory platforms to inform the system's design and features.
2. **System Design (Late May - Early June 2025):** Following the literature review, the focus shifts to designing the system architecture. This includes creating the system architecture diagrams, use case diagrams, data flow diagrams, and the database schema (ER Diagram). This phase establishes the technical blueprint for the entire application.
3. **Backend Development (June 2025):** This phase involves the implementation of the server-side logic. Key activities include setting up the SQLite database, developing the user registration and authentication modules, creating the personalized health recommendation engine, and integrating the notification system.
4. **Frontend Development (Late June - Mid-July 2025):** With the backend in place, development of the user interface begins. Using the Streamlit framework, this phase focuses on creating a responsive, intuitive, and user-friendly interface that allows users to register, manage their profiles, interact with the chatbot, and view their health plans.
5. **Integration and Testing (Mid-July - Late July 2025):** This critical phase involves integrating the frontend and backend components to ensure they function together seamlessly. Rigorous testing is conducted to identify and resolve any bugs, validate the accuracy of the health recommendations, and assess the overall usability and performance of the system.
6. **Finalization and Documentation (August 2025):** The final phase involves preparing the project for completion. This includes finalizing all features, compiling comprehensive project documentation, and preparing the final report and presentation materials.

# Chapter 9: Conclusion

## Conclusion

The Smart Health Assistant project successfully addresses the critical and growing need for healthcare solutions that are accessible, personalized, and preventive. By leveraging modern technologies like Artificial Intelligence, Natural Language Processing, and cloud-based services, the system empowers individuals to take a proactive and informed role in managing their health.

The platform offers a unique combination of personalized health plans, AI-driven symptom analysis, and timely preventive reminders, effectively bridging the gap between patients and essential healthcare resources. Its design, which targets a wide demographic by segmenting recommendations based on age and gender, ensures that the advice provided is both inclusive and relevant.

Furthermore, the system's modular architecture is built for scalability and security, providing a solid foundation for future enhancements such as the integration of data from wearable health devices and the implementation of more advanced predictive analytics. The successful completion of this project not only contributes to individual well-being but also aligns with broader public health goals by promoting early health interventions and helping to reduce the overall burden on traditional healthcare systems.

In essence, the Smart Health Assistant serves as a powerful demonstration of how innovative digital solutions can significantly improve lives, support healthcare ecosystems, and foster healthier communities.

# Reference

Bass, L., Clements, P., & Kazman, R. (2012). Software architecture in practice (3rd ed.).

Addison-Wesley Professional.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877–1901.

https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a- Paper.pdf

Owens, M. (2010). The definitive guide to SQLite. Apress.

Patel, S., Patel, N., & Merchant, S. (2022). Artificial intelligence in health care: Applications and challenges. Healthcare Analytics, 2(1), 100015. https://doi.org/10.1016/j.health.2022.100015

Project Management Institute. (2017). A guide to the project management body of knowledge (PMBOK® Guide) (6th ed.). Project Management Institute.

Streamlit Team. (2022). Streamlit documentation. https://docs.streamlit.io/

Statista. (2024). Number of mHealth app users worldwide from 2016 to 2024 (in billions). https://[www.statista.com/statistics/779919/global-mhealth-users/](http://www.statista.com/statistics/779919/global-mhealth-users/)

Topol, E. (2019). Deep medicine: How artificial intelligence can make healthcare human again.

Basic Books.

VanderPlas, J. (2016). Python data science handbook: Essential tools for working with data.

O'Reilly Media.

World Health Organization. (2021). Global strategy on digital health 2020–2025. https://[www.who.int/publications/i/item/9789240020924](http://www.who.int/publications/i/item/9789240020924)

Lee, J., & Yoon, W. (2021). Emerging trends in telemedicine and digital health solutions: A review. Journal of Medical Internet Research, 23(5), e24908. https://doi.org/10.2196/24908