Kalasalingam Academy of Research and Education

Code Crunch 2024
(National Level Hackathon)

Low-Level Design

on

STREE KAVACH

Team members: T Saisree, J Manojna, A Bhavya sri, k Lasya

Table of Contents

1. Introduction

1.1 Scope of the document

This Low-Level Design (LLD) document serves as the blueprint for designing and implementing a chatbot that handles health and legal issues. It outlines the essential components, interactions, and processes required to develop a robust and secure chatbot system. The document covers various aspects, including client-side and server-side validation, component design, configuration settings, and interfaces with other system components.

Key functionalities covered:

Client-Side Validation:

- **Input Validation:** Ensures that the user input meets the required format (no special characters, proper syntax) before sending it to the server.
- **Real-Time Feedback:** Provides immediate feedback to users if their input is incorrect or incomplete (missing required fields).

Server-Side Validation

- **Deep Input Validation:** Verifies the authenticity and completeness of user input on the server side, including more complex validation checks.

Interaction with External APIs:

- **Health Information Retrieval:** Integrates with external health APIs to provide users with medical advice, symptom checking, and general health information.

Database Management:

- **User Data Storage:** Securely stores user data, including interaction history, preferences, and other relevant information.
- **Querying and Updating:** Manages database operations to retrieve and update user data as needed during interactions.

Response Generation:

- **Dynamic Response Creation:** Generates contextually relevant responses based on the user's input, combining data from external APIs, database queries, and business logic**.**

Error Handling and Recovery:

- **Graceful Error Handling:** Provides meaningful error messages to users when something goes wrong and suggests corrective actions.

Security and Authentication:

- **User Authentication:** Implements authentication mechanisms to ensure that only authorized users can access certain features or data within the chatbot.
- **Access Control:** Restricts access to sensitive data and functions based on user roles and permissions.

User Interface (UI) Integration:

- **Seamless UI Interaction:** Ensures smooth communication between the chatbot and the user interface, providing a responsive and intuitive user experience.

1.2 Intended audience

This document is intended for a diverse group of stakeholders involved in the design, development, deployment, and maintenance of the chatbot system. Each audience group can use this document to understand different aspects of the system and contribute to its success.

Software Developers

- **Responsibilities:** Implementation of the chatbot system, including backend logic, API integration, database management, and UI/UX design.
- **Focus Areas:** Detailed flowcharts, sequence diagrams, validation processes, API interactions, database schema, and configuration settings.
- **Purpose:** To provide guidance on how to develop, test, and integrate various components of the chatbot system.

System Architects:

- Designing the overall system architecture, ensuring scalability, security, and performance.
- System architecture design, component interactions, security measures, data flow, and integration points.
- To validate and refine the system architecture, ensuring it meets the project's technical and business requirements.

Quality Assurance

- Testing the chatbot system to ensure it meets the defined requirements and is free of defects.

Security Analysts

- Ensuring that the chatbot system is secure, with robust protection against threats and vulnerabilities.
- Security measures, authentication, encryption, access control, and audit logging.

Legal and Compliance:

- Ensuring that the chatbot system complies with relevant laws, regulations, and industry standards, especially in the context of handling legal and health-related information.

Business Analysts :

- Analyzing business requirements and translating them into technical specifications for the chatbot system.

1.3 System overview

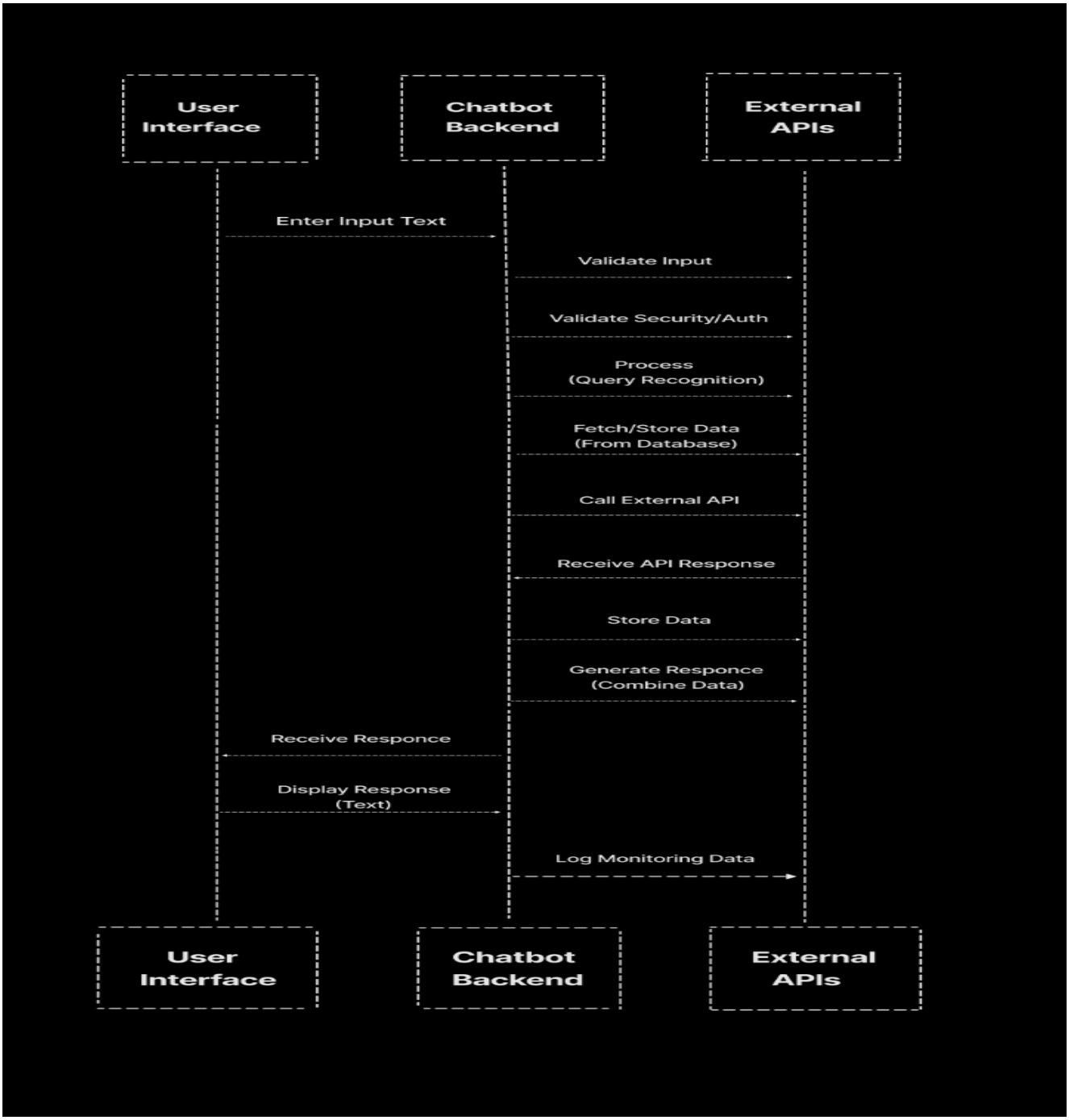
- The chatbot system designed to handle health and legal issues is a comprehensive solution that combines secure data management, and robust integration with external services. This system aims to provide users with

accurate, timely, and contextually relevant information
in response to their queries while ensuring compliance with legal and regulatory standards.

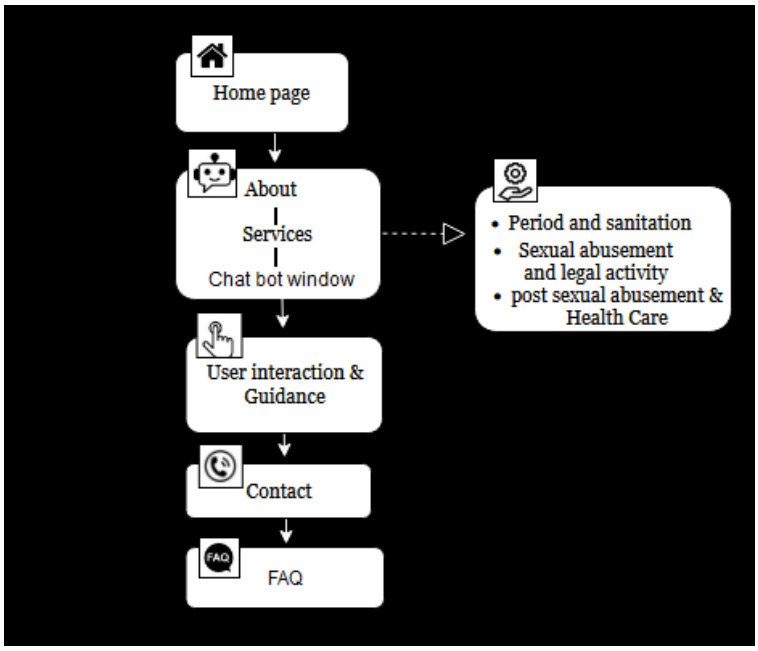
- **Chatbot Backend:** The core logic and processing engine of the system, validation, API calls, and response generation. It connects the user interface with the necessary data sources and services.
- **External APIs:** The chatbot integrates with external health and legal services APIs to retrieve up-to-date information, legal references, and health guidelines**.**
- **Database:** A secure storage system for user data, interaction logs, configuration settings, and other necessary data.
- **Data Flow:** The system handles data in a secure and efficient manner, with information flowing from the user interface to the backend, where it is processed and possibly enriched with data from external APIs or the database, before being returned to the user.
- **Intent Recognition and Entity Extraction:** The NLP engine interprets user queries, identifying the intent (e.g., asking for legal advice or health recommendations) and extracting relevant entities (e.g., names, dates, symptoms).
- **Real-Time and Server-Side Validation:** The system ensures that user inputs are validated both on the client-side for format and syntax, and on the server-side for deeper checks, such as legal compliance or health data accuracy.
- **API Integration:** The system can dynamically call external health and legal APIs to fetch the latest information or provide services like symptom checking, legal document analysis, or legal advice.
- **Secure Data Handling:** All user data is encrypted and securely stored, with strict access control mechanisms in place. Sensitive information is protected according to legal and regulatory standards.
- **Response Generation:** Based on the processed input and data retrieved from various sources, the system generates and delivers accurate, contextually relevant responses to the user.
- **Logging and Monitoring:** Continuous logging and monitoring ensure the system operates reliably, with any issues being quickly identified and addressed.

2. Low Level System Design

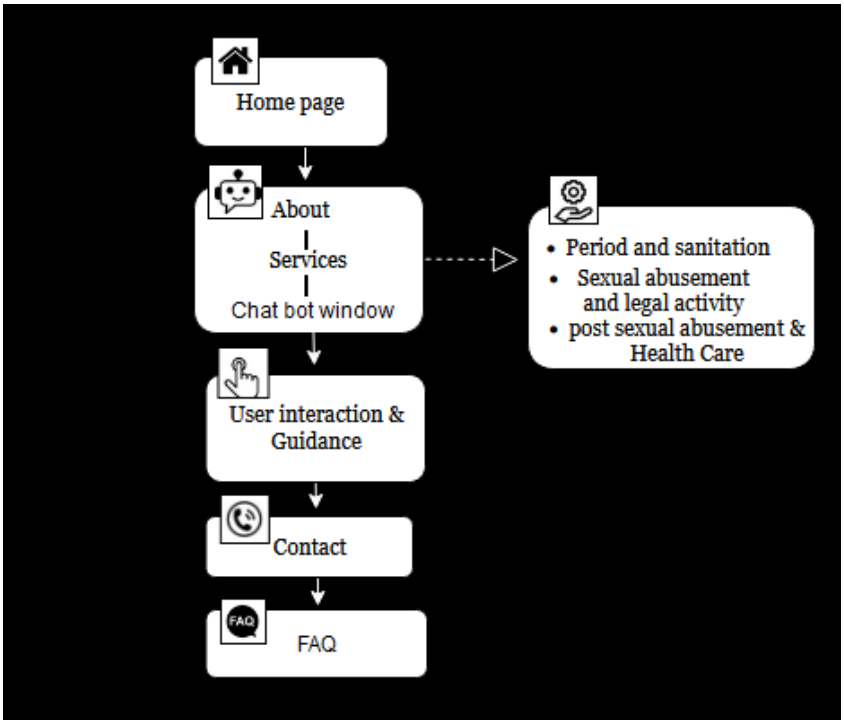
1.1 Sequence Diagram



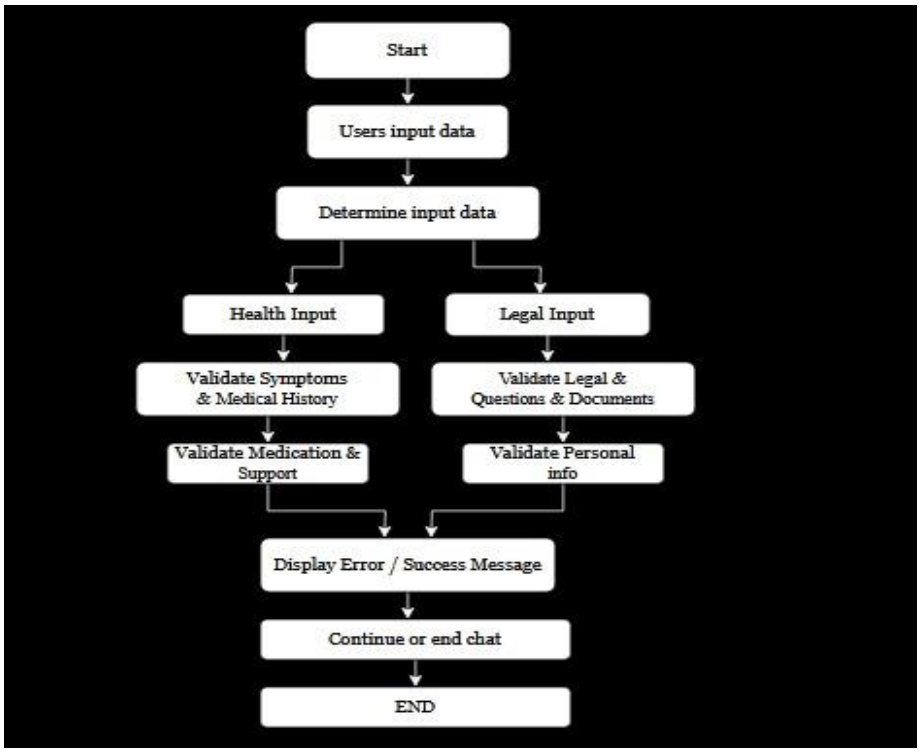
1.2 Navigation Flow/UI Implementation



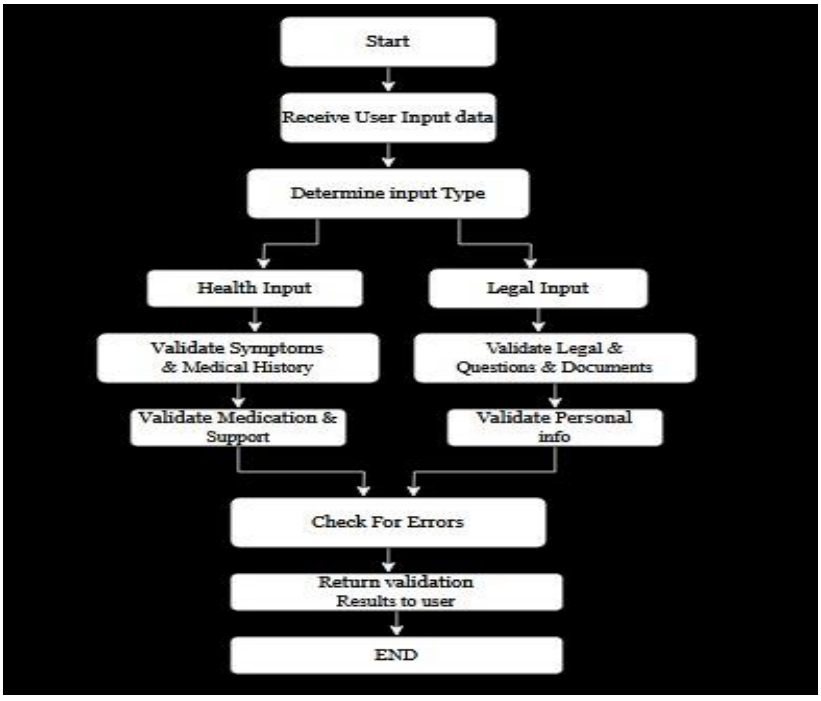
1.3 Screen Validations, Defaults and Attributes



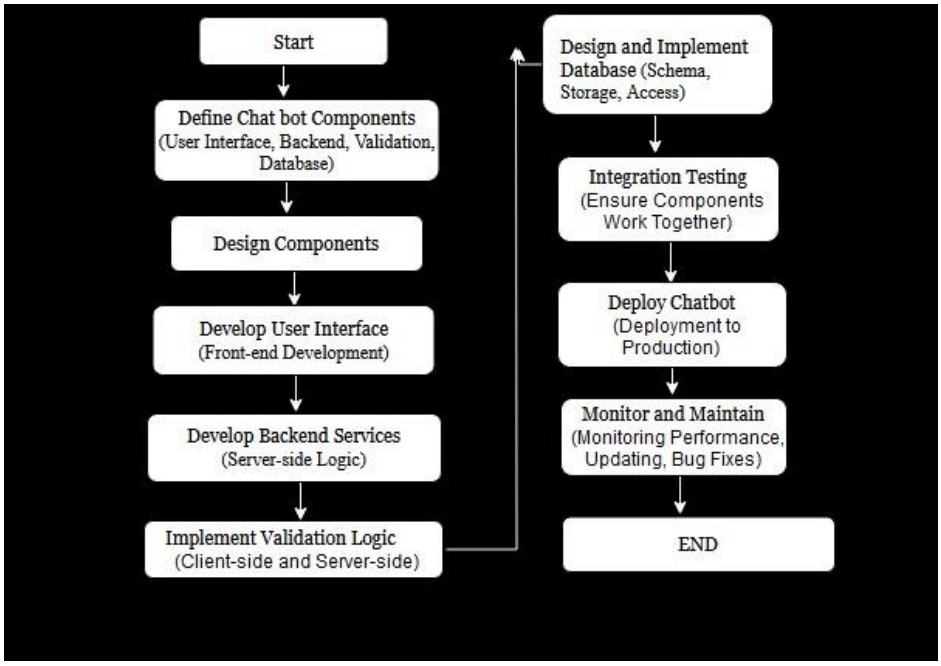
1.4 Client-Side Validation Implementation



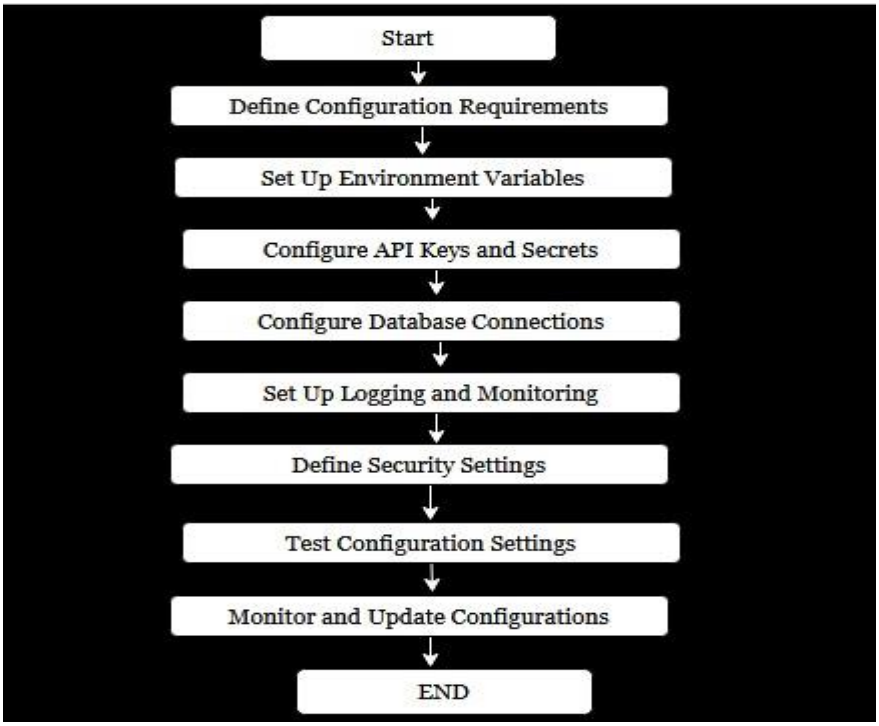
1.5 Server-Side Validation Implementation



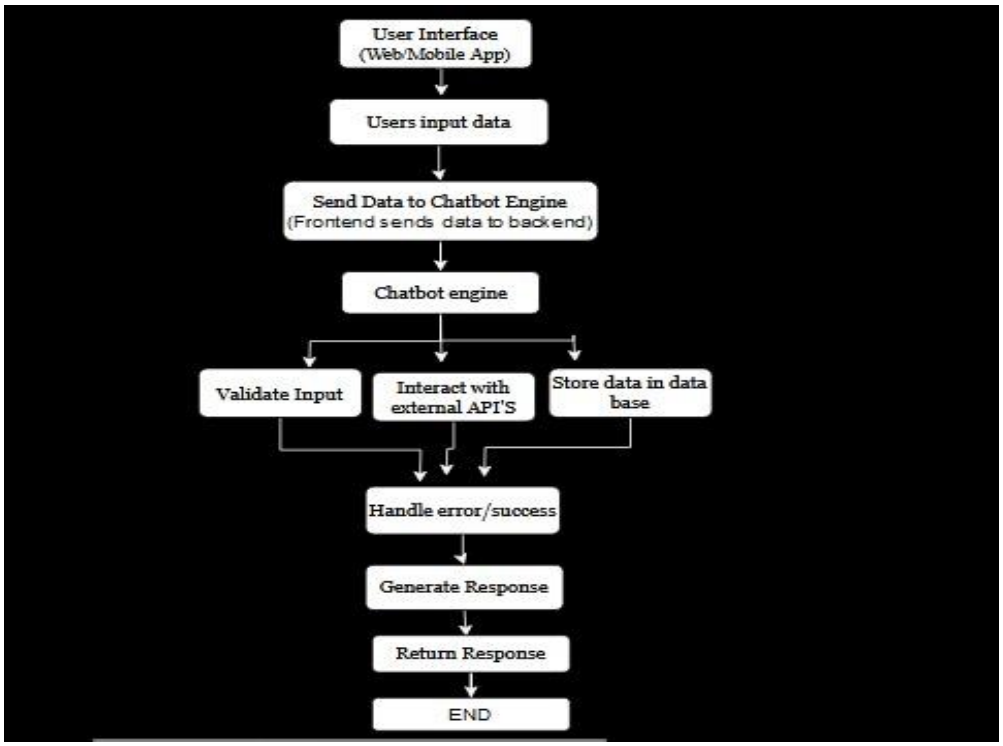
1.6 Components Design Implementation



1.7 Configurations/Settings



1.8 Interfaces to other components



3. Data Design

1.1 List of Key Schemas/Tables in database

The database design for the chatbot system encompasses several key schemas and tables essential for managing user interactions, integrating external data, and maintaining system configurations.

1. **User Management:** is facilitated through the `Users` table, which stores essential profile information such as user IDs, encrypted passwords, email addresses, and roles. Accompanying this is the `User Preferences` table, which records individual user settings and preferences, such as language and theme choices.
2. **Interaction Logging:** The `User Interactions` table tracks detailed logs of user inputs and chatbot responses, including timestamps and identified intents. The `Error Logs` table captures system errors with detailed messages and stack traces to aid in troubleshooting.
3. **External Data Integration**: is supported by the `Health Data` and `Legal Data` tables, which store data retrieved from external APIs related to health and legal information, respectively. These tables include fields for data source, type, content, and timestamps.
4. **System Configuration:** Is managed through the `System Settings` table, which holds global configuration settings, and the `Feature Toggles` table, which controls the

activation of various system features. These tables ensure that system settings and feature states are easily adjustable.

5. **Security and Access Control:** Are handled by the `Roles` and `User Roles` tables. The `Roles` table defines various user roles and associated permissions, while the `User Roles` table links users to their respective roles, ensuring appropriate access controls and permissions.

This structured approach to database design ensures that the chatbot system is both functional and secure, effectively supporting its intended operations and maintaining robust performance.

1.2 Details of access levels on key tables in scope

1. **Users Table**:

Read Access:

Admins: Full access to view all user profiles and details.

Users: View their own profile information.

Write Access:

Admins: Can modify user profiles, including updating roles or resetting passwords.

Users: Can update their own profile information, such as email or password**.**

Administrative Access:

Admins: Full control to add, modify, or delete user records.

Health Data Table

- **Admins:** View all health data records.
- **System Components:** Read access for integrating and processing health data.
- **Write Access:** Write access for storing health data retrieved from external APIs.
- **Administrative Access:** Full control over health data records, including deletion or modification if necessary.

Legal Data Table:

- **Admins:** View all legal data records.
- **System Components:** Read access for integrating and processing legal data.
- **Write Access:** Write access for storing legal data retrieved from external APIs.
- **Administrative Access:** Full control over legal data records, including deletion or modification if necessary.

1.3 Key design considerations in data design

This section delves into the essential principles shaping our data architecture, ensuring efficiency, security, and scalability for our innovative tool.

1. **Normalization**:
   - Minimize data redundancy: By organizing data into well-defined tables with clear relationships, we eliminate duplicate information, reducing storage requirements and improving data integrity.

- Enhance data consistency: Normalization simplifies data updates and ensures consistency across different parts of the system.
- Optimize query performance: Normalized structures enable efficient data retrieval and manipulation, leading to faster response times and improved user experience.

2. **Scalability**: Anticipate future growth: We design the database with flexibility in mind, allowing for seamless expansion as the user base and data volume increase.
   - Utilize horizontal scaling: By leveraging technologies like distributed databases or sharing, we can distribute data across multiple servers, enhancing processing power and handling larger datasets.
   - Choose appropriate data types: Selecting data types that efficiently store the expected data size and range optimizes storage utilization and query performance.
3. **Security**:
   - Protect user privacy: We implement robust security measures like encryption, access control, and data anonymization to safeguard sensitive user information.
   - Comply with regulations: Adherence to relevant data privacy regulations like GDPR ensures user trust and protects against potential legal ramifications.
   - Prevent unauthorized access: Implementing strong authentication and authorization mechanisms restricts access to authorized users and prevents unauthorized data manipulation.
4. **Performance**:
   - Optimize query performance: We design efficient database queries that retrieve and process data quickly, minimizing wait times and ensuring responsiveness.
   - Utilize indexing: Indexing key fields in tables significantly speeds up data retrieval, especially for frequently used queries.
   - Cache frequently accessed data: Caching frequently accessed data in memory reduces database load and improves overall system performance.
5. **Data Integrity**:
   - Enforce data validation: Implementing validation rules at different levels (client-side, server-side) ensures data accuracy and consistency during input and processing.
   - Define data constraints: Setting constraints on data types, formats, and relationships prevents invalid data from entering the system.
   - Regular data cleansing: Periodically review and cleanse the data to eliminate errors, inconsistencies, and outdated information.

4. Details of other frameworks being used

1.1 Session Management

Session management is a crucial aspect of the chatbot system, particularly when dealing with sensitive health and legal information. Effective session management ensures that user interactions are maintained consistently, securely, and efficiently across multiple exchanges. Let's explore its key advantages:

- **Unique Identification:** Each user session is assigned a unique identifier (Session ID) that the system uses to track the session across different requests.
- **Session Storage**: For temporary session data that does not need to be persisted, in-memory databases like Redis can be used to store session information. Sessions should have an expiration time after which they are invalidated, ensuring that stale sessions do not remain active indefinitely.
- **Enhanced Security** Regenerate Session IDs upon user login and periodically during the session to mitigate session fixation attacks. Implement monitoring to detect and respond to unusual activity, such as multiple login attempts from different locations.
- **Session Data Optimization:** Store only essential information in the session to reduce memory usage and improve performance.

1.2 Caching

Caching is a technique used to store frequently accessed data temporarily in a high-speed storage layer, enabling faster data retrieval and reducing the load on backend systems. In a chatbot system that handles health and legal queries, caching can significantly enhance performance, improve user experience, and reduce response times.

- **Improve Performance**: By storing frequently accessed data in a cache, the system can retrieve this data much faster than from a database or an external API.
- **Reduce Latency:** Caching reduces the time taken to fetch data, resulting in quicker responses to user queries.
- **Decrease Server Load:** Caching offloads repetitive data retrieval tasks from the main database or APIs, reducing the overall load on these systems.
- **Enhance Scalability:** With efficient caching, the system can handle more concurrent users without degradation in performance.
- **Redis**: Leveraging Redis as a cache offers additional benefits alongside Stream lit's .
- **Data Persistence**: While Memcached primarily focuses on in-memory caching, Redis offers optional persistence options if needed. This ensures data isn't lost in case of server restarts or failures.
- **Complex Data Structures**: If our caching needs involve storing more complex data structures like lists or sets, Redis supports them efficiently, offering greater flexibility than Memcached's key-value pairs.
- **Integration with Stream Processing**: If our project incorporates real-time data analysis or dynamic features, Redis' stream processing capabilities could be valuable for efficient data handling alongside its caching functionality.

1.3 Additional Tools and Technologies:

Frontend: While Stream lit manages the frontend, you might mention any custom components or front-end libraries used (e.g., JavaScript frameworks).

Backend: If a separate backend framework is used, describe its role and integration.

Redis An in-memory data structure store used as a database, cache, and message broker. Known for its high performance and scalability.
Programming Language: Python 3 is standard, but mention any other languages used for specific tasks.
Packages: List all relevant Python packages used for functionality beyond Stream lit's core capabilities.

5. **Unit Testing**

Unit testing is a crucial part of the development process, ensuring that each individual component of the chatbot system functions correctly in isolation. By thoroughly testing each unit, developers can identify and resolve issues early in the development cycle, leading to a more reliable and maintainable system.
Purpose: Identify and fix bugs at an early stage, before they can cause more significant issues in the integration or system testing phases. Provide a safety net for developers when making changes to the codebase, ensuring that modifications do not introduce new bugs.

Client-Side Validation

- Input Format Validation: Test the functions that check for correct input formats.
- Real-Time Feedback Mechanisms: Verify that the system provides immediate feedback to users for incorrect or incomplete input, preventing submission of invalid data.

Server-Side Validation

- **Data Integrity Checks:** Ensure that server-side validation functions correctly validate user input for data integrity, such as verifying legal document formats or medical codes.
- **Security Validation:** Test that the system correctly handles and rejects potentially malicious input (SQL injection attempts, cross-site scripting).
- **Complex Validation Logic:** Validate that the server-side logic correctly processes and validates complex data inputs, such as multi-field forms or nested data structures.

API Integration

- **API Call Success:** Test that the system can successfully call external APIs and receive the expected responses.
- **Error Handling:** Verify that the system handles API errors (timeouts, invalid responses) gracefully, providing appropriate fallback responses or error messages.
- **Response Parsing:** Ensure that the system correctly parses and processes data received from external APIs, converting it into a format that can be used by the chatbot.

Response Generation

- **Dynamic Response Creation:** Test that the system generates accurate and contextually relevant responses based on the user's input and the data retrieved from APIs or databases.
- **Multi-Modal Responses:** Verify that the system can generate responses in different formats depending on the interaction mode.
- **Edge Cases:** Test how the response generation logic handles edge cases, such as missing or incomplete data from external sources.

Database Operations

- **Data Storage and Retrieval**: Test functions responsible for storing and retrieving user data, ensuring data integrity and correctness.

- **Query Performance:** Verify that database queries are efficient and return the expected results within a reasonable time frame.
- **Error Handling:** Ensure that the system correctly handles database errors, such as connection issues or query failures, and implements appropriate fallback mechanisms.

Security and Authentication

- **Authentication Logic:** Test the authentication functions to ensure that only authorized users can access protected features or data.
- **Encryption:** Test that sensitive data is properly encrypted during storage and transmission, ensuring data confidentiality.

6. Key notes

- The chatbot system is designed to address both health and legal inquiries, providing users with accurate, reliable, and timely information through a single platform.
- Both client-side and server-side validations are implemented to ensure data accuracy, security, and compliance with relevant regulations. This helps prevent incorrect or malicious input from affecting the system.
- The chatbot integrates with external health and legal APIs to retrieve up-to-date information, enhancing its capabilities to provide users with the latest advice and guidelines.
- The chatbot is accessible across various platforms and devices, providing a seamless user experience whether accessed via a web interface, mobile app, or other messaging platforms.
- The system is designed with maintainability in mind, allowing for regular updates, patches, and improvements to ensure it remains secure, efficient, and compliant over time.

7. Reference

- *Date, C. J. (2004). "An Introduction to Database Systems."* - This book provides a comprehensive introduction to database concepts, design, and management.
- *Harrington, J. L. (2016). "Database Design: Practical Guide for DBA's."* - A practical guide to database design with a focus on best practices and implementation strategies.
- *Stallings, W. (2021). "Computer Security: Principles and Practice."* - This book covers key concepts in computer security, including access control mechanisms and data protection strategies.
- *EST API Tutorial* - An online resource for understanding RESTful API design and implementation practices.
- *Fowler, M. (2018). "Refactoring: Improving the Design of Existing Code."* - Discusses refactoring techniques and maintaining code quality, including the role of unit testing.