



SOFTWARE DESIGN SPECIFICATION

Artificial Intelligence  
QuickChat - Chatbot

|  |  |  |  |
| --- | --- | --- | --- |
| **Created By:** | EDGAR PINTO | **Approved By:** | Harshada Topale |
| **Created On:** | 31-08-2024 | **Approved On:** | DD-MMM-YYYY |

**INDEX**

[**1** **PURPOSE** 2](#_Toc142418236)

[**2** **PROJECT SCOPE** 2](#_Toc142418237)

[**3** **SYSTEM OVERVIEW** 2](#_Toc142418238)

[**4** **DESIGN CONSIDERATIONS** 2](#_Toc142418239)

[4.1 Requirements 3](#_Toc142418240)

[4.2 Assumptions 3](#_Toc142418241)

[4.3 Dependencies 3](#_Toc142418242)

[**5** **SYSTEM ARCHITECTURE** 3](#_Toc142418243)

[5.1 Architectural Strategies 4](#_Toc142418244)

[5.2 Structure & Relationships 4](#_Toc142418245)

[**6** **DETAILED DESCRIPTION OF COMPONENTS** 4](#_Toc142418246)

[**7** **INTEGRATION** 5](#_Toc142418247)

[**8** **APPENDICES** 1](#_Toc142418248)

[8.1 Appendix A – Detailed Description of Components 1](#_Toc142418249)

# **PURPOSE**

The purpose of this project is to create a chatbot that efficiently handles queries from users across multiple channels like Facebook, LinkedIn, WhatsApp, and SMS. This chatbot aims to reduce customer support costs and improve response times for Cloud Counselage Pvt Ltd and the Industry Academia Community (IAC). By integrating FAQs and leveraging AI technologies such as speech recognition and generative AI, the chatbot will offer a seamless user experience. The project will focus on delivering a high-quality, user-friendly solution that effectively supports students and freshers in the IAC by providing instant support and information.

# **PROJECT SCOPE**

The scope of QuickChat includes its distinct features, benefits, and limitations. QuickChat's unique features allow it to deliver efficient, multi-channel customer support by using Node.js for the backend and React for the frontend. The system automates the handling of frequently asked questions (FAQs) and can be enhanced with AI-driven speech recognition and generative AI for more interactive and natural conversations. This helps solve the problem of delayed or fragmented customer support, making QuickChat a valuable tool for reducing support costs and improving response times. However, its effectiveness relies on the accuracy of input data and the sophistication of the AI models used.

# **SYSTEM OVERVIEW**

QuickChat is composed of several key components and subsystems, each playing a vital role in delivering a seamless customer support experience. Below is an outline of its main components:

1. **User Interface (Frontend)**: The frontend is built using React, providing a responsive and user-friendly interface where users can interact with the chatbot. It supports text input and potentially voice input, displaying responses in real-time. The design is optimized for accessibility and cross-device compatibility.
2. **Backend Server**: The backend, developed with Node.js, serves as the core processing unit of QuickChat. It manages the communication between the frontend and other subsystems, processes user inputs, and retrieves responses. The backend also handles routing, session management, and integration with external APIs.
3. **OpenAI API Integration**: QuickChat leverages the OpenAI API for its natural language processing and AI capabilities. The backend uses an OPENAI\_API\_KEY to authenticate and interact with the API, enabling the chatbot to generate responses and engage in more complex interactions. The ASSISTANT\_ID is used to identify and manage different instances of the chatbot, ensuring personalized and consistent user experiences.
4. **Integration Layer**: The integration layer connects QuickChat with external platforms and tools, allowing it to function within various ecosystems. This includes handling API calls, data exchanges, and ensuring seamless operation across different channels.
5. **Analytics and Reporting**: This subsystem collects and analyses data on user interactions, system performance, and response accuracy. It provides valuable insights for improving the chatbot’s efficiency and user satisfaction over time.

# **DESIGN CONSIDERATIONS**

## Requirements:

* The system must support multi-channel communication and offer a responsive, user-friendly interface.
* QuickChat should efficiently handle user interactions across platforms like Facebook, LinkedIn, WhatsApp, and SMS.
* The backend needs to manage communication between the frontend and the integrated platforms.
* Security measures, such as encryption and authentication, are necessary to protect user data.

## Assumptions

* The APIs of the platforms QuickChat integrates with will remain stable and accessible.
* Users will have basic familiarity with the platforms they use to interact with QuickChat.
* The development team will have continuous access to necessary tools and resources.
* The frontend and backend components will be compatible with the targeted platforms.

## Dependencies

* QuickChat’s functionality relies on the availability and stability of third-party platform APIs.
* The success of the project depends on the secure handling of user data and integration processes.
* The frontend's performance is tied to the responsiveness and support of the integrated platforms.
* Availability of development and security tools is crucial for successful implementation.

# **SYSTEM ARCHITECTURE**

The software system architecture refers to the logical organization of a distributed system into software components. It defines how components of a software system are assembled, their relationship and communication between them. It serves as a blueprint for software application and development basis for developer team. An effective architecture serves as the conceptual glue that holds every phase of the project together for all of its stakeholders, enabling agility, time and cost savings, and early identification of design risks.

**Operational Architecture Characteristics:**

* Availability
* Performance
* Reliability
* Low fault tolerance
* Scalability

**Structural Architecture Characteristics:**

* Configurability
* Extensibility
* Supportability
* Portability
* Maintainability

**Cross-Cutting Architecture Characteristics:**

* Accessibility
* Security
* Usability
* Privacy
* Feasibility

## Architectural Strategies

1. **Frontend:**

* **Technology:** React
* **Purpose:** User interface and interaction
* **Components:** UI elements, state management, and client-side logic

1. **Backend:**

* **Technology:** Node.js
* **Purpose:** Server-side logic and API handling
* **Components:** RESTful APIs, server-side processing, and data management

1. **AI Integration:**

* **Technologies:** Speech recognition, generative AI
* **Purpose:** Enhance interactions with AI-driven features
* **Components:** AI models, integration endpoints, and processing logic

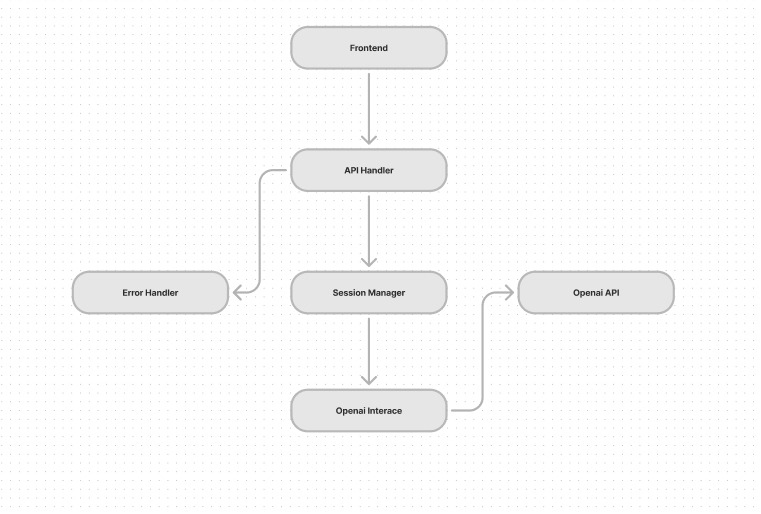
1. **Communication Layer:**

* **Purpose:** Facilitates data exchange between frontend and backend
* **Components:** API endpoints, data serialization/deserialization

## Structure & Relationships

The structure and relationships between the components of your chatbot system:

* **Frontend (React)**: The user interface that interacts with users.
* **API Handler**: Manages communication between the frontend and backend.
* **Session Manager**: Tracks user sessions and maintains context.
* **OpenAI Interface**: Connects with the OpenAI API to generate responses.
* **Error Handler**: Manages and logs errors.
* **OpenAI API**: External service used to generate dynamic responses.

****

# **DETAILED DESCRIPTION OF COMPONENTS**

|  |  |
| --- | --- |
| **Identification** | **QuickChat** Chatbot Backend |
| **Type** | A module, a subprogram, a form, a data file, a control procedure, a class, etc. |
| **Purpose** | * **Function**: Manages API requests and responses, controls session flow, and integrates with the OpenAI API for query handling. * **Performance Requirements**: Real-time processing of user queries with minimal latency. Should support multiple simultaneous interactions. * **Derived Requirements**: Secure API communication and robust error management. |
| **Subordinates** | **Internal Structure**:   * **API Handler**: Manages incoming/outgoing data. * **Session Manager**: Tracks user interactions across channels. * **OpenAI Interface**: Communicates with the API to fetch responses. * **Error Handler**: Logs and manages error conditions. |
| **Dependencies** | **Component Dependencies**:   * **Frontend (React)**: Interface between backend and user interactions. * **OpenAI API**: For generating dynamic responses. * **Communication Channels**: Handles message flow and authentication.   **Interaction Details**:   * **Timing**: Real-time query processing. * **Execution Order**: API Handler -> Session Manager -> OpenAI Interface -> Response sent. * **Data Sharing**: User inputs, session tokens, and API responses. |
| **Interfaces** | **External Interfaces**:   * **OpenAI API**: Requires API key and assistant ID for generating responses. * **Frontend (React)**: Communicates through RESTful APIs.   **Internal Interfaces**:   * **Session Manager**: Connects with API Handler to manage user sessions.   **UI Components**: User interactions are handled across different communication methods. |
| **Resources** | **Software**: OpenAI API, Node.js, relevant npm libraries, express.js |
| **Processing** | * **User Query Handling**: Real-time query processing. * **Session Tracking**: Manages user sessions. *  **Error Handling**: Logs and manages issues. |
| **Data** | JSON-based session and API data. |

# **INTEGRATIONS**

# **AI Models Integration:**

* + **Tools/Services:** OpenAI API
  + **Purpose:** Integrate AI models for generative responses and natural language processing.
  + **Details:** The backend will use the OpenAI API to process user queries and generate responses**. This involves:**
    - **API Calls: Making requests to the OpenAI API with user input. Response Handling: Processing the API responses and integrating them into the application's data flow.**
    - **AI Model Management: Updating or fine-tuning models as needed to improve performance.**

1. **Frontend-Backend Communication:**
   * **Tools/Services: RESTful APIs**
   * **Purpose: Facilitate data exchange between the frontend and backend.**
   * **Details: The frontend (React) will:**
     + **API Calls: Send HTTP requests to the backend (Node.js) for various operations such as fetching FAQs, sending messages, and updating user information.**
     + **Endpoints: Defined RESTful endpoints will manage operations like /getchat, /sendMessage, and /updateUser.**
     + **Data Handling: Serialize and deserialize data for communication between the client and server.**

# **APPENDICES**

## Appendix A – Detailed Description of Components

|  |  |
| --- | --- |
| **Identification** | **ChatInterface** |
| **Type** | Class/Form/ |
| **Purpose** | The Chat Interface allows users to interact with the chatbot and access various support functions. It handles user inputs and displays responses. |
| **Subordinates** | Chat History Screen |
| **Dependencies** | Links to the following screens:   * Chat History Screen |
| **Interfaces** | The interface includes text input fields for user messages and a display area for chat responses. It features buttons for accessing chat history and settings. |
| **Resources** | **History Storage:** Chat history is stored and managed through the OpenAI Assistant. The system retrieves and displays historical chat data via API interactions with OpenAI. |
| **Processing** | Handles real-time message exchange with the backend. User inputs are sent to the backend, processed by the AI models, and responses are displayed in the chat window. Historical chat data is accessed from OpenAI Assistant. |
| **Data** | The data includes user messages, chatbot responses, and timestamps. This data is used to maintain and display chat history stored in OpenAI Assistant. |