**Hybrid AI-Human Architecture for Real-Time Customer Support: Leveraging Retrieval-Augmented Generation**

**Abstract**

This paper presents the design and implementation of a real-time, embeddable customer service system that integrates Retrieval-Augmented Generation (RAG) with human-in-the-loop (HITL) escalation. The system addresses common limitations in traditional customer support, such as slow response times and inconsistent service, by combining intelligent automation with timely human intervention. Built using FastAPI, ReactJS, PostgreSQL, and Centrifugo for WebSocket communication, the architecture supports low-latency interaction and seamless transitions between bot and human agents. At its core, the system leverages OpenAI’s GPT-3.5-turbo model to generate responses informed by user-provided domain-specific intent files.

To evaluate the effectiveness of the system, we conducted controlled testing using metrics such as average response time, session resolution rate, escalation latency, and accuracy of chatbot replies. Results showed that 71% of user queries were successfully handled by the chatbot alone, with an average response time under 3 seconds and human agent intervention occurring in less than 3 minutes for escalated cases. These outcomes suggest that the proposed system offers a scalable and intelligent alternative for real-time customer engagement, particularly in environments where domain-specific support and rapid escalation are essential.

**Keyword:** Retrieval-Augmented Generation (RAG), Customer Service Automation, Human-in-the-Loop (HITL), Chatbot Architecture, Large Language Models (LLMs), Real-Time AI Systems

**1. INTRODUCTION**

In the digital era, customer service has emerged as a critical differentiator for business success, especially in highly competitive and consumer-driven markets. Traditional support systems, typically reliant on phone calls, emails, and manual human agents, often fail to meet modern expectations for immediacy and personalization. Long wait times, inconsistent service quality, and the inability to scale during peak demand periods have left customers dissatisfied and businesses under pressure to innovate [1].

To address these limitations, businesses have increasingly turned to artificial intelligence (AI), especially AI-powered chatbots. These systems leverage natural language processing (NLP) to simulate human conversation and automate responses to customer inquiries. However, despite advances in AI, many chatbot systems still suffer from shortcomings such as limited contextual understanding, poor handling of multi-turn dialogue, and inflexible, rule-based logic [2]. These limitations can result in miscommunication, reduced customer trust, and the eventual need for human intervention.

The emergence of large language models (LLMs), such as OpenAI’s GPT-3 and GPT-4, has marked a turning point in conversational AI. These models are capable of generating contextually relevant and coherent responses by learning from vast amounts of structured and unstructured data [3]. When integrated with Retrieval-Augmented Generation (RAG), LLMs can enhance response accuracy by drawing upon external sources of domain-specific knowledge during conversation generation. This makes RAG an effective solution for customer service scenarios that demand both fluency and factual accuracy [4].

Despite the promise of such technologies, research indicates that AI alone cannot fully replace the need for human judgment, especially in handling emotionally sensitive or complex customer queries [5]. Human-in-the-loop (HITL) systems, which allow seamless escalation from chatbot to human agents, have emerged as a practical approach to overcoming these challenges. By combining automation with human oversight, HITL systems ensure that customers receive not only instant replies but also empathetic and accurate resolutions when automation reaches its limits.

This paper presents the design and implementation of a real-time, embeddable customer service system that integrates RAG-based chatbot architecture with dynamic human escalation. The solution is built on a modular web-based framework using FastAPI, ReactJS, and Centrifugo for WebSocket-based communication. It allows businesses to embed a customizable AI assistant into their digital platforms and configure it using domain-specific intent files. When the chatbot encounters queries beyond its capability, the system triggers an escalation mechanism that alerts human agents in real time via NovuHQ notifications.

In addition to its core functionality, the system features an analytics dashboard for performance monitoring, response categorization, and conversational summaries. It supports multiple integration formats, including iframe, web component, and shareable URLs, making it suitable for diverse business contexts, from small enterprises to large-scale service providers.

**2. LITERATURE REVIEW**

The rapid evolution of artificial intelligence (AI) and natural language processing (NLP) has significantly influenced customer service systems, enabling businesses to automate responses, improve scalability, and enhance user satisfaction. This section reviews existing literature on AI-powered chatbots, Retrieval-Augmented Generation (RAG), and human-in-the-loop (HITL) mechanisms, identifying their strengths, limitations, and opportunities for innovation.

### ****2.1 Evolution of Chatbots in Customer Service****

Early chatbot systems, such as ELIZA [9] and PARRY [4], were rule-based and relied on pattern matching to simulate conversation. While pioneering, these systems lacked contextual memory and adaptability. Subsequent developments introduced AI-powered chatbots capable of basic intent recognition using decision trees and keyword extraction [10].

In recent years, the integration of natural language processing (NLP) and machine learning has significantly advanced chatbot capabilities. AI-powered platforms such as Google Dialogflow, IBM Watson Assistant, and Microsoft Bot Framework now support intent classification, entity recognition, and conversational flow design [10]. These platforms offer improved response quality but often fall short in handling dynamic, multi-turn conversations or domain-specific queries without substantial manual training and fine-tuning.

**2.2 Large Language Models and RAG in Dialogue Systems**

The emergence of large language models (LLMs) such as GPT-3 and GPT-4 has reshaped conversational AI by enabling more fluent, contextually coherent responses. These models, trained on massive corpora, can generate text, summarize content, and answer questions with high degrees of linguistic accuracy [11]. However, their generative nature can also lead to hallucinations or factually incorrect outputs.

To mitigate this, Retrieval-Augmented Generation (RAG) has been introduced as a hybrid approach that combines document retrieval with generation. Lewis et al. [4] propose RAG as a method that retrieves relevant information from an external knowledge base and conditions the generative model on it, thereby improving factual accuracy and contextual grounding. This is especially valuable in customer service, where responses must be consistent with company policies, FAQs, or service documentation.

Several studies have demonstrated the effectiveness of Retrieval-Augmented Generation (RAG) in dialogue systems. For instance, Agrawal et al. [6] proposed Beyond-RAG, a real-time conversational framework that first classifies the user’s intent, checks for matching FAQs, and conditionally invokes RAG only when necessary. Their system achieved sub-2-second response times in live customer service settings, demonstrating that RAG-based architectures can support low-latency, production-grade deployments. However, despite such advancements, few implementations focus on embeddable, self-configurable platforms suitable for integration into external web environments.2.3 Human-in-the-Loop (HITL) Approaches in Customer Support. Despite advances in LLMs and RAG, studies emphasize that fully autonomous systems often fail to adequately handle emotionally sensitive or high-stakes situations [5], [12]. Human-in-the-loop (HITL) models have thus emerged as a pragmatic solution, where automated systems manage routine interactions and escalate complex issues to human agents.

**2.3 Real-Time Systems and Embeddable Architectures**

Real-time chatbot systems require robust backend architectures capable of handling concurrent sessions and asynchronous message processing. A multi-layered architecture, typically involving the user interface, logic engine, and data integration layer, is critical for ensuring scalability and low-latency response. Technologies such as WebSockets and asynchronous task queues like Celery are commonly used to facilitate real-time communication and background processing [4]. Additionally, the adoption of API**-**first development paradigms has enabled the creation of embeddable, self-contained chatbot components that can be integrated directly into third-party platforms with minimal configuration.

While much of the existing literature focuses on building scalable chatbots, relatively few studies address solutions that are both embeddable and configurable using domain-specific intent files. This is particularly limiting for small and medium-sized enterprises (SMEs) that require flexible, low-code AI support tools. Emerging tools like LangChain offer frameworks for integrating retrieval-augmented generation (RAG) in production environments [8], but few implementations have emphasized real-time interaction or simplified configuration for business users. This work addresses that gap by introducing a system that blends RAG-based AI with human-in-the-loop escalation, packaged in a modular, real-time architecture suitable for diverse customer service contexts.

**2.4 Research Gap**

Although numerous works have explored AI chatbots, RAG models, and HITL strategies independently, few have addressed their integration into a unified, embeddable customer service framework. While Agrawal *et al.* [6] introduced Beyond-RAG, a real-time RAG-based conversational system used in contact centres, and Afzal *et al.* [13] explored a RAG implementation with human-in-the-loop evaluation, these systems lack the modular embeddability and domain-specific configuration necessary for SMEs. Similarly, the recent SymbioticRAG framework [14] emphasizes human-LLM collaboration for document intelligence but is tailored toward document exploration rather than integrated customer support. The lack of solutions that are both low-latency, easy to embed, and customizable via intent files motivates our work.

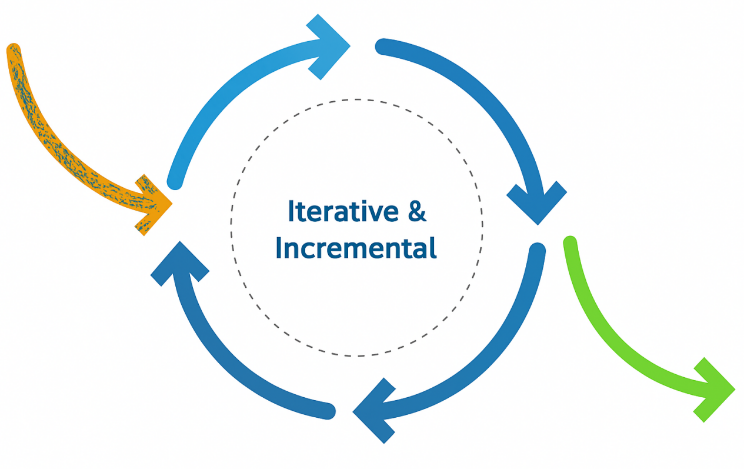
Our paper addresses this gap by presenting an architecture that combines real-time RAG, configurable HITL escalation, and analytics-enabled embeddability, packaged in a platform-agnostic form suitable for rapid deployment in the service domain.

**3. METHODOLOGY**

This section outlines the methodological approach adopted in designing and implementing the real-time, embeddable Retrieval-Augmented Generation (RAG) system for customer service with human-in-the-loop (HITL) escalation. The methodology spans the selection of development models, system components, and architectural decisions aimed at achieving the project objectives, namely, real-time processing, intelligent query handling, and seamless human agent escalation.

**3.1 Development Approach**

The **Iterative Software Development Model** was adopted due to its flexibility and emphasis on incremental refinement. Each iteration involved the design, implementation, and evaluation of specific components, such as the chatbot engine, the intent-based retrieval mechanism, and the escalation framework [15]. Feedback was continuously incorporated to enhance system usability and performance.

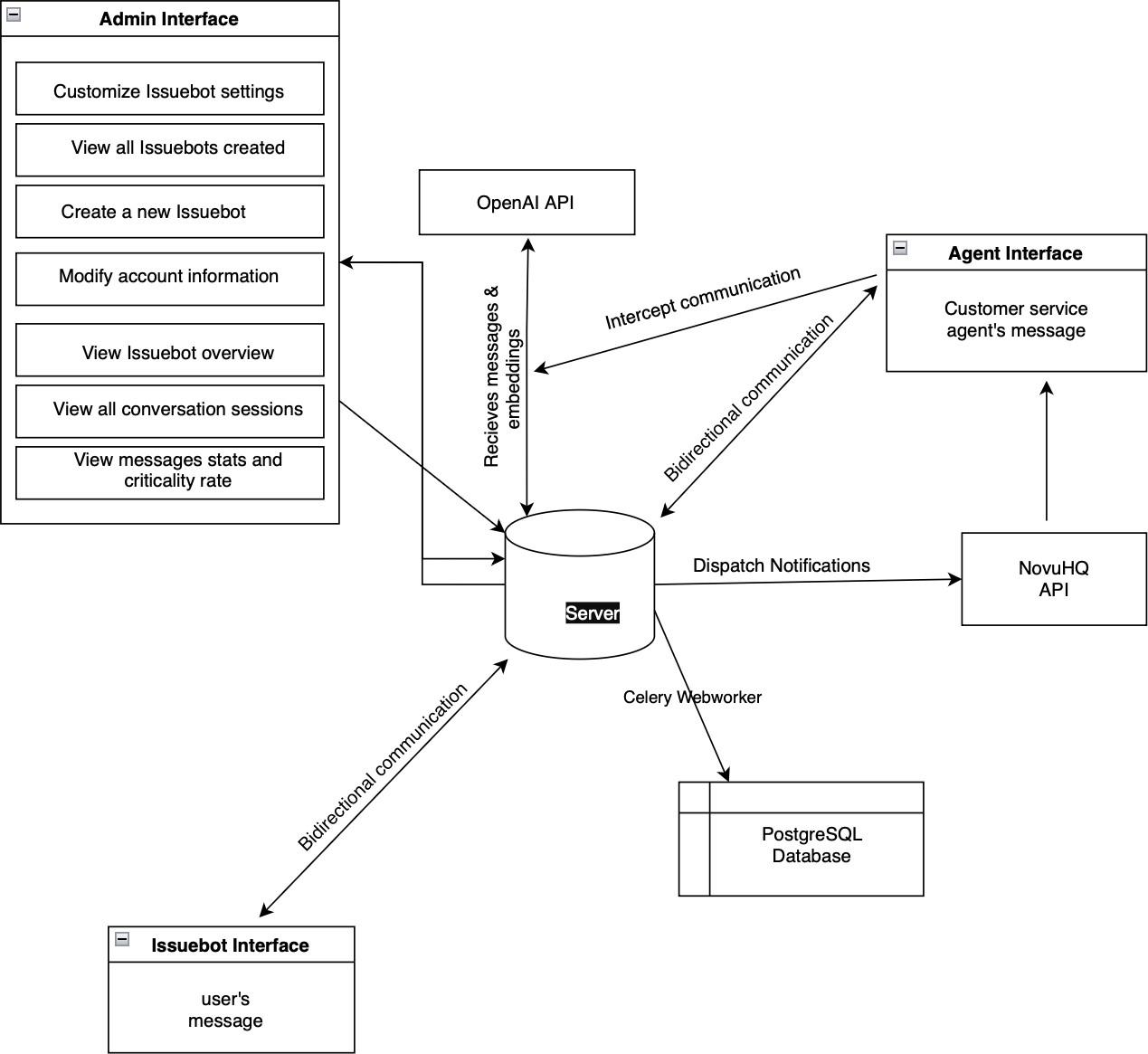


**Figure 1**. Iterative Software Development Cycle

**3.2 System Workflow Overview**

The system consists of several integrated components that handle user input, process retrieval and generation tasks, and escalate unresolved issues to human agents. The core flow involves:

1. A customer initiates a query through the web-based chatbot interface.
2. The backend receives the message and generates an embedding of the user input.
3. Using vector similarity, the system retrieves relevant content from a user-uploaded **intent file.**
4. A RAG-enabled prompt is formed using the retrieved content and passed to the **OpenAI GPT API**, which returns a response.
5. If the chatbot cannot confidently respond or the user explicitly requests human assistance, the message is escalated to a human agent using **Centrifugo** (WebSocket-based real-time communication) and **NovuHQ** notifications.
6. The agent seamlessly joins the session, and both interaction data and system metrics are logged in the PostgreSQL database.

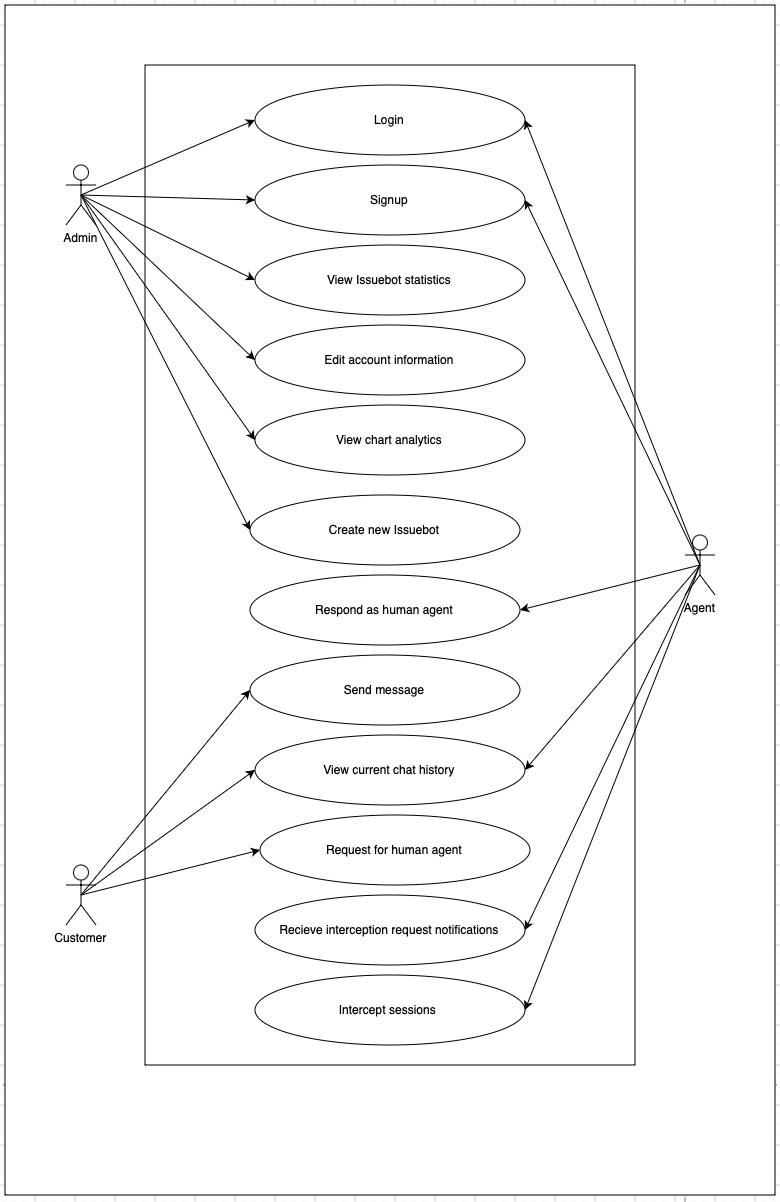


**Figure 2.** High-Level Architecture of the System

**3.3 Use Case and Functional Design**

The system supports three primary actors:

1. **End Users** (Customers) interact with the chatbot, request assistance, and receive responses.
2. **Human Agents** receive escalation alerts, join ongoing chats, and intervene when necessary.
3. **Administrators** upload and manage intent files, configure escalation rules, and monitor system performance.



**Figure 3.** Actor-Based Use Case Representation

**3.4 Sequence of Operations**

The communication and task flow are captured in a **sequence diagram**, showing the interactions among the frontend, backend, database, RAG engine, and escalation services. It emphasizes the decision points for automated vs. human response routing.

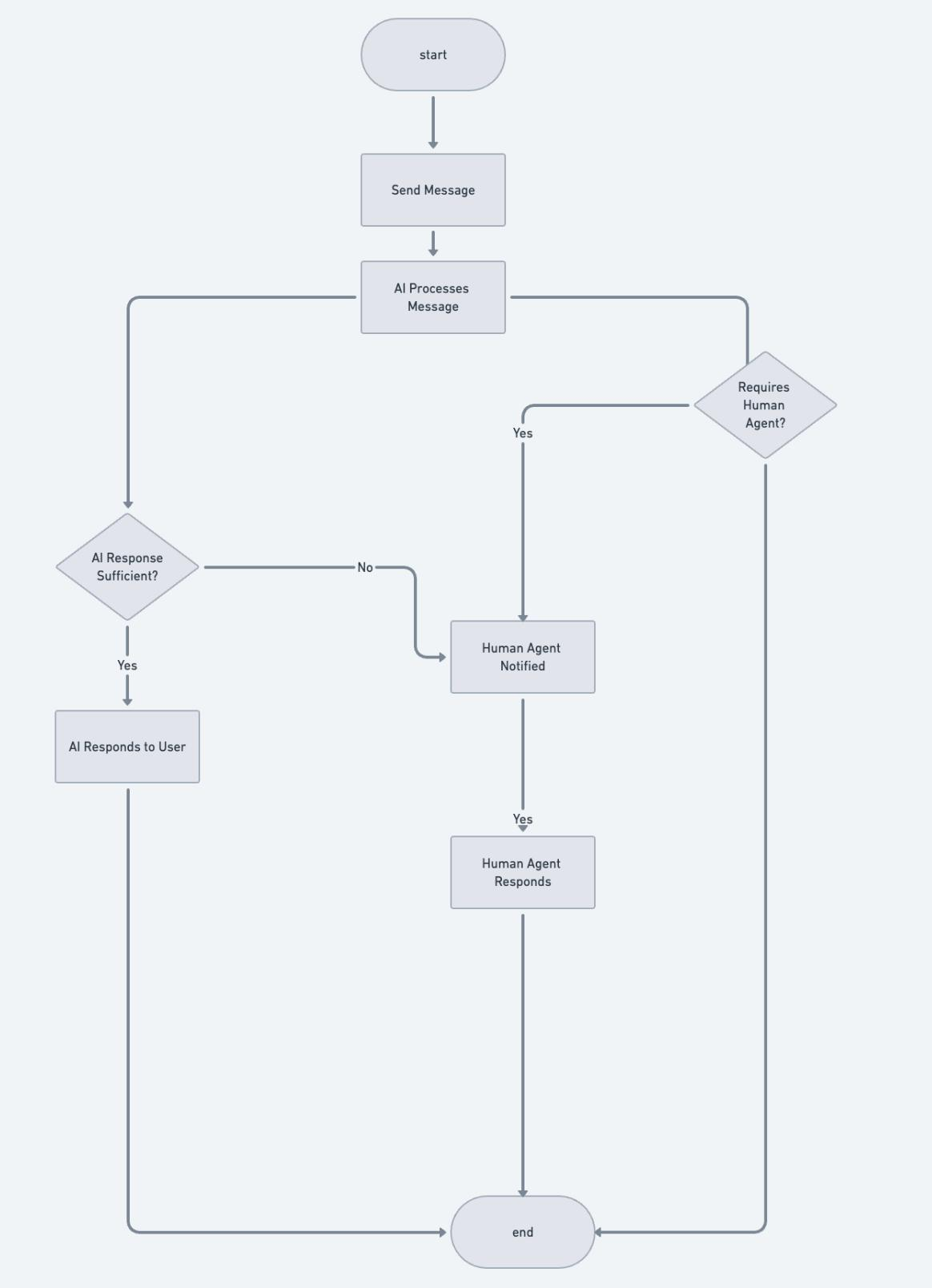
### 

**Figure 4.** System Message and Escalation Sequence

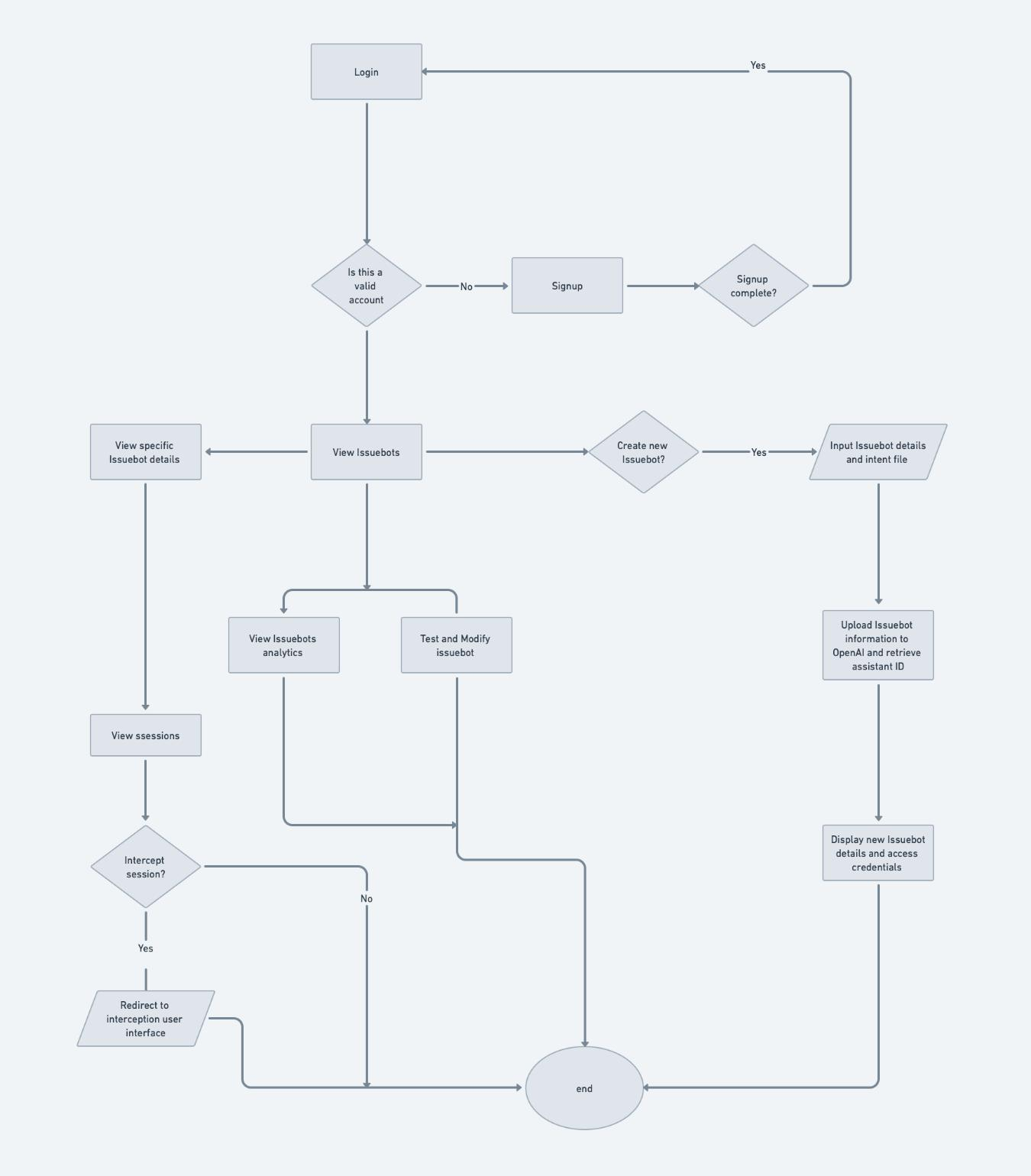
**3.5 User Experience and Backend Logic Flow**

To capture both user and system behaviours, two diagrams are proposed:

1. **User Flow Diagram:** Captures typical customer interactions, from initiating a query to receiving assistance.
2. **System Workflow Diagram:** Represents how backend components like FastAPI, Celery, and Centrifugo handle requests asynchronously and communicate with external APIs.



**Figure 5.** User flow diagram



**Figure 6.** System Processing Flowchart

**3.6 Data Modelling and Storage**

The backend of the proposed system is supported by a relational data model implemented in **PostgreSQL**, designed to manage real-time chatbot sessions, track escalation events, and provide analytical insights. The core entities include users, chat sessions, messages, intent files, escalation logs, and analytics. Each user, whether a customer, human agent, or administrator, is stored with a role-based profile. When a user initiates a conversation, a session is created and linked to all messages exchanged within that session. These messages are recorded with sender type and timestamps for traceability. If a conversation requires human intervention, an escalation log is created to capture details such as the responding agent, response time, and escalation type (manual or automatic). Meanwhile, businesses upload intent files, which are indexed and used during the retrieval step of the RAG workflow. To enable performance monitoring, interaction data is aggregated in the analytics table, recording metrics like average response time, resolution duration, and escalation frequency. The model is normalized for integrity and indexed for speed, supporting high-performance query operations in real-time environments.

**3.7 Tools and Technologies**

**Table 1.** Tools, Technologies, and Purpose

|  |  |  |
| --- | --- | --- |
| **Component** | **Technology** | **Purpose** |
| Frontend | ReactJS, Vite | Lightweight, interactive user interface |
| Backend | FastAPI | Core API logic and business rule processing |
| Chat Engine | OpenAI GPT (via API) | Natural language understanding and generation |
| Retrieval Mechanism | Vector similarity search (e.g., FAISS or in-memory) | Matches user input with relevant intent data |
| WebSocket Layer | Centrifugo | Enables real-time messaging |
| Notifications | NovuHQ | Triggers alerts for escalation |
| Task Queue | Celery | Handles background jobs and long-running tasks |
| Database | PostgreSQL | Stores structured data and chat histories |

**4. IMPLEMENTATION**

The implementation phase focused on translating the system design into a functional, web-based application that supports real-time, intelligent customer interaction. The system was developed using a modular, API-first architecture to ensure flexibility, embeddability, and integration with external platforms.

**4.1 Technology Stack and Tooling**

The backend was implemented using **FastAPI**, a modern Python framework well-suited for asynchronous APIs. It handles request routing, session management, and integration with external services. **Celery** was used for executing background jobs such as logging, session analysis, and periodic updates. The real-time communication layer was built using **Centrifugo**, which enables WebSocket connections between the frontend and backend. For notifications during escalation, **NovuHQ** was integrated to push alerts to available agents.

The frontend was built using **ReactJS** and bundled with **Vite**, offering a fast, lightweight interface that supports chat interactions, bot testing, analytics dashboards, and administration tools. User authentication was managed using Google OAuth and password-based login, with session handling secured by token-based authorization.

**4.2 Chatbot and Retrieval-Augmented Generation Integration**

The chatbot functionality was powered by **OpenAI's GPT-3.5-turbo model**, with RAG (Retrieval-Augmented Generation) used to enhance accuracy. When a user submits a query, the system:

1. Generates an embedding of the query using OpenAI’s embedding API,
2. Searches a pre-computed vector space derived from a user-uploaded intent file (using FAISS or in-memory search),
3. Combines the retrieved context with the query,
4. Constructs a prompt and sends it to the OpenAI chat API for response generation.

The response is then returned to the frontend in real-time. If the chatbot determines, based on predefined rules or confidence levels, that the query is outside its scope, it flags the message for escalation.

**4.3 Human Escalation Mechanism**

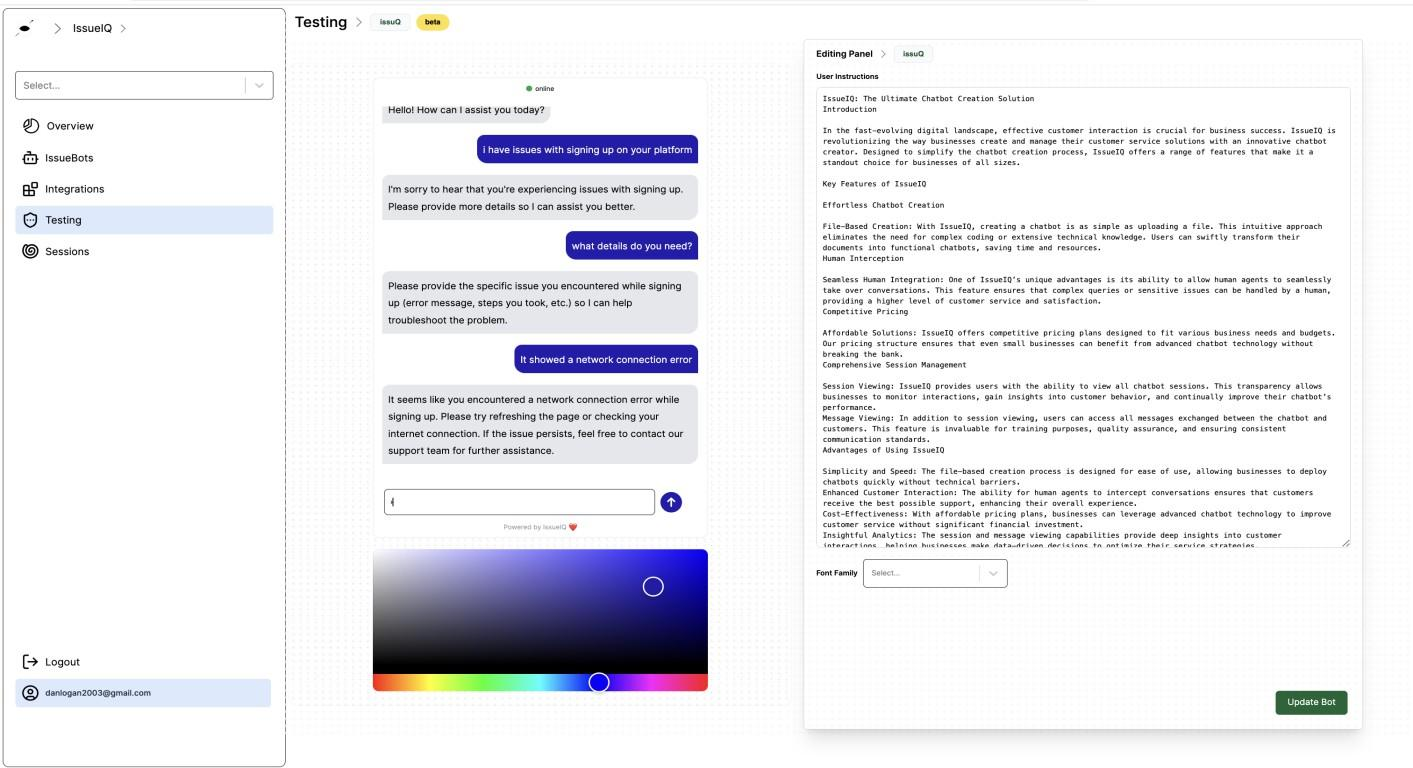
Upon escalation, the system uses **NovuHQ** to send real-time notifications to available agents. The agent receives the session ID and full message context and can immediately join the conversation through the agent interface. A seamless transition is achieved via Centrifugo’s WebSocket layer, which synchronizes the chat window without reloading or disrupting the user experience. Agents can respond directly, resolve the issue, or escalate further as needed. The conversation can also be returned to the chatbot after the issue is clarified.

**4.4 Key Functional Modules**

The system's frontend was developed using ReactJS and structured into distinct, modular components to support ease of use, responsiveness, and maintainability. Each module serves a specific purpose in the chatbot management and interaction lifecycle. The system emphasizes embeddability, real-time feedback, and flexible human-agent integration. This section provides a breakdown of the core components, their interaction logic, and the engineering considerations behind their implementation.

**4.4.1 Chat Interface**

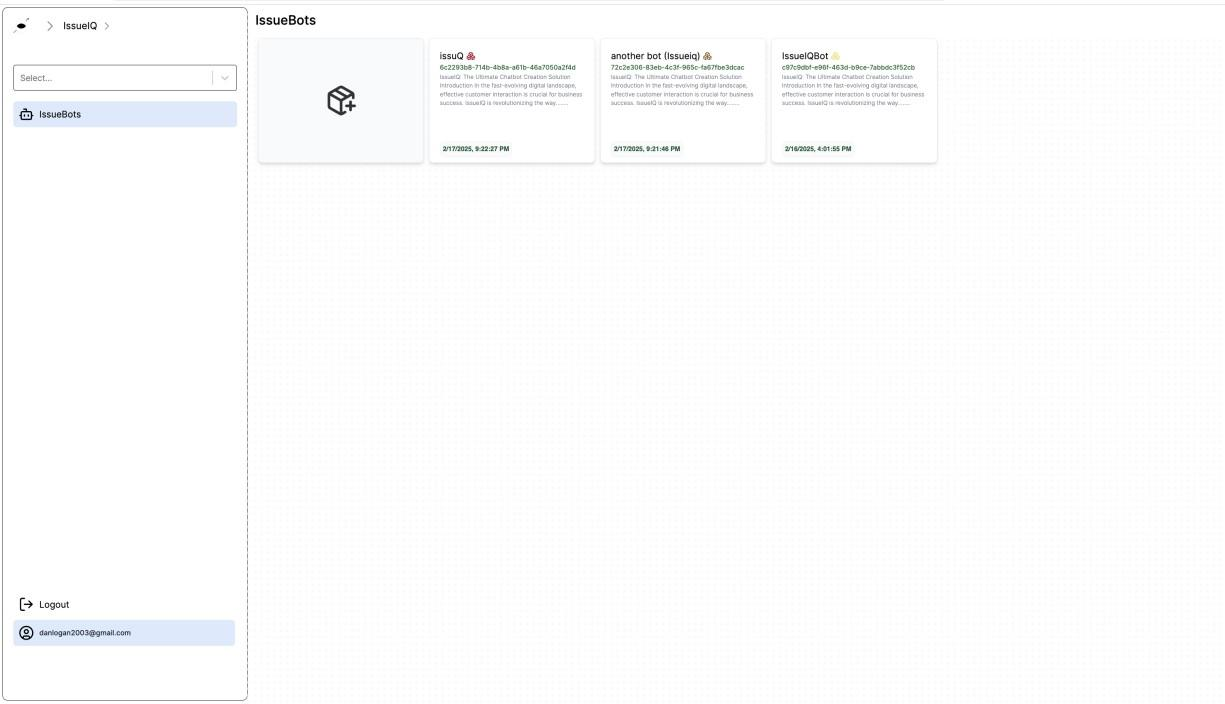
The Chat Interface is the central point of interaction for end users. It allows users to input queries, receive responses from the chatbot, and communicate with a human agent when necessary. It supports real-time messaging using Centrifugo WebSockets, which eliminates the need for periodic polling and ensures low-latency communication. Each message is timestamped and labeled based on the sender type, whether from the user, the chatbot, or a human agent. The interface itself is built with a lightweight ReactJS component and bundled using Vite, enabling fast rendering and compatibility with common embedding strategies such as iframes and web components. When a user sends a message, it is routed to the FastAPI backend, where the system processes it using the RAG engine to generate a response. If an escalation condition is detected, a flag is set on the active session, and the agent module is notified in real time via WebSocket updates.



**Figure 7.** Chat interface and editing panel

**4.4.2 Bot Management Panel (IssueBot Control Center)**

This component enables users (especially admin roles) to configure chatbot instances, manage intent files, and define escalation parameters. It allows administrators to upload domain-specific intent files in structured .txt or .md formats. Upon upload, the system initiates a background Celery task that generates embeddings and stores the resulting vector index for retrieval during chatbot interactions. Administrators can then preview the chatbot’s behavior through an integrated testing window that simulates conversations using the uploaded knowledge. This design empowers non-technical users to adapt the chatbot to their specific domain without requiring model retraining, making the system more accessible and configurable for diverse organizational needs.

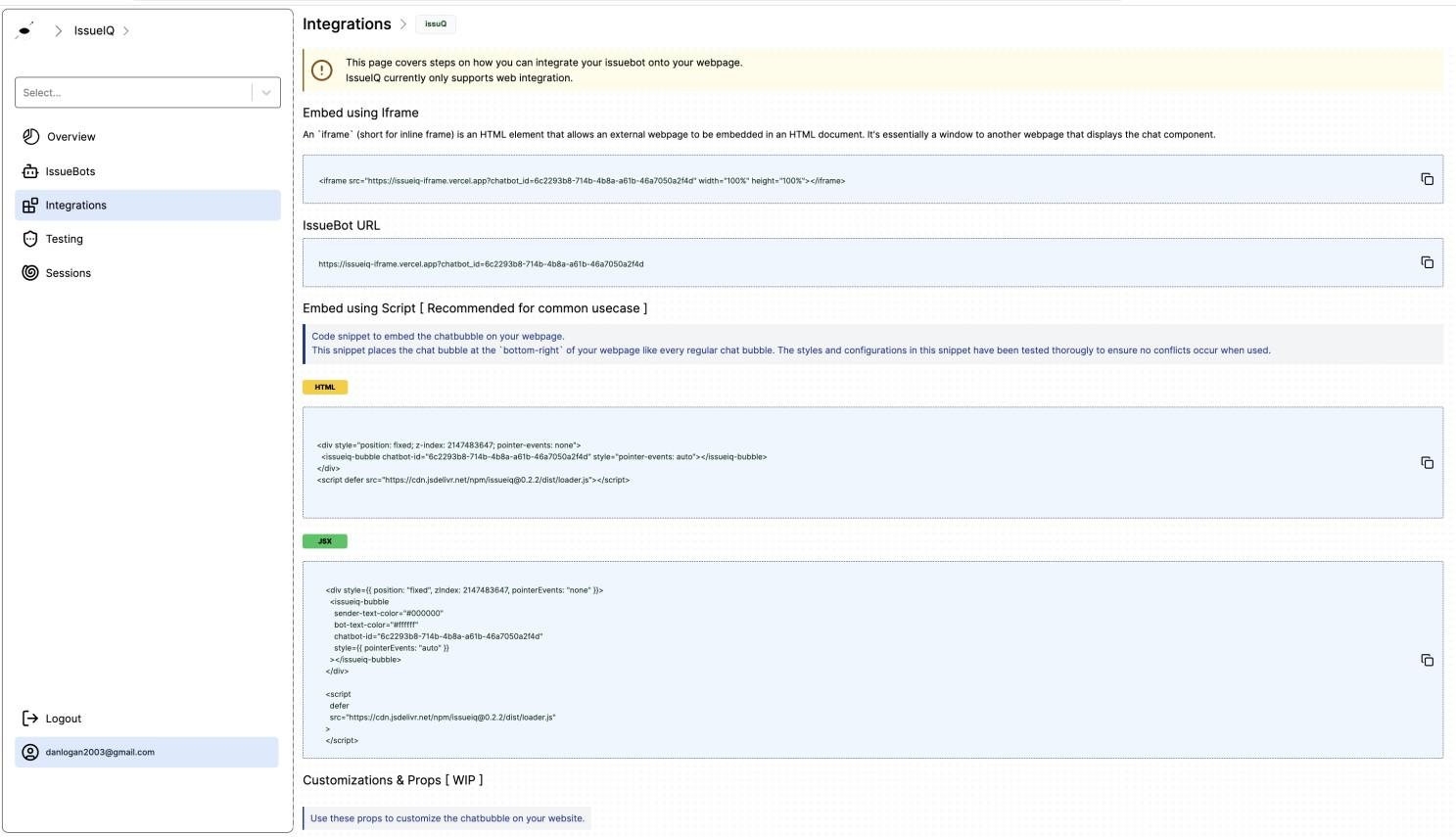


**Figure 8.** Bot management panel

**4.4.3 Integrations Page**

The system provides a dedicated **Integrations Page** designed to support flexible deployment of the IssueBot chatbot across a variety of platforms, catering to both technical and non-technical users. This page outlines three primary methods through which the chatbot can be embedded or shared, allowing organizations to choose the integration strategy that best aligns with their environment.

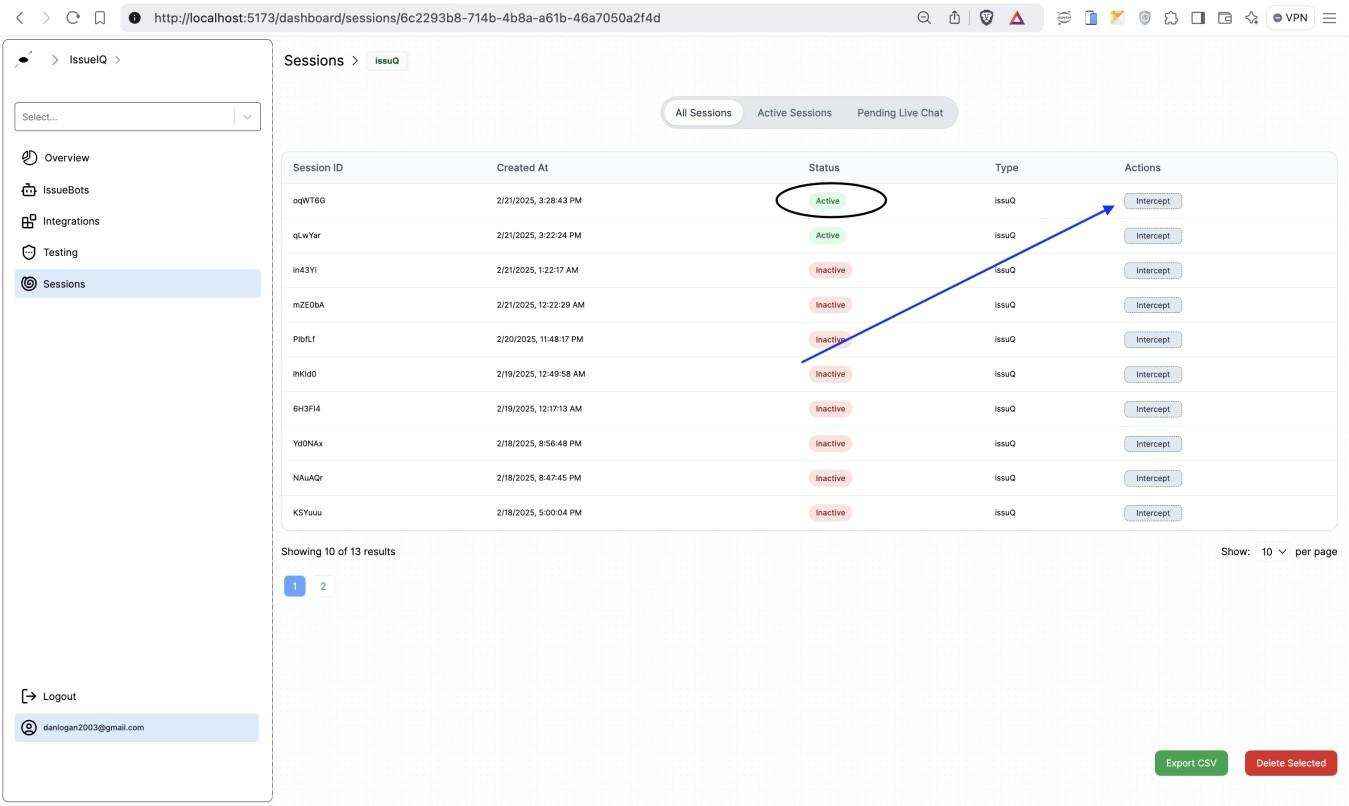
1. **Embedding via an <**iframe**>**, which enables users to incorporate the chatbot directly into a webpage. This approach is ideal for quick integration, allowing seamless user interaction within existing websites without requiring significant technical effort.
2. Sharing **a generic web URL**. This link leads to a hosted version of the chatbot interface, enabling end users to access the bot directly without embedding it into a webpage. It is particularly useful for organizations that want to distribute the chatbot through email, social media, or mobile platforms.
3. Embedding **via a web component script**: This technique uses a custom JavaScript snippet to render the chatbot as a reusable web component. It supports modern frontend environments using HTML, JSX, or TSX, making it well-suited for integration within single-page applications or component-based frameworks. This method offers advanced customization options and greater control over styling and behavior.



**Figure 9.** Showing the Integrations Page

**4.4.4 Session Handling Page**

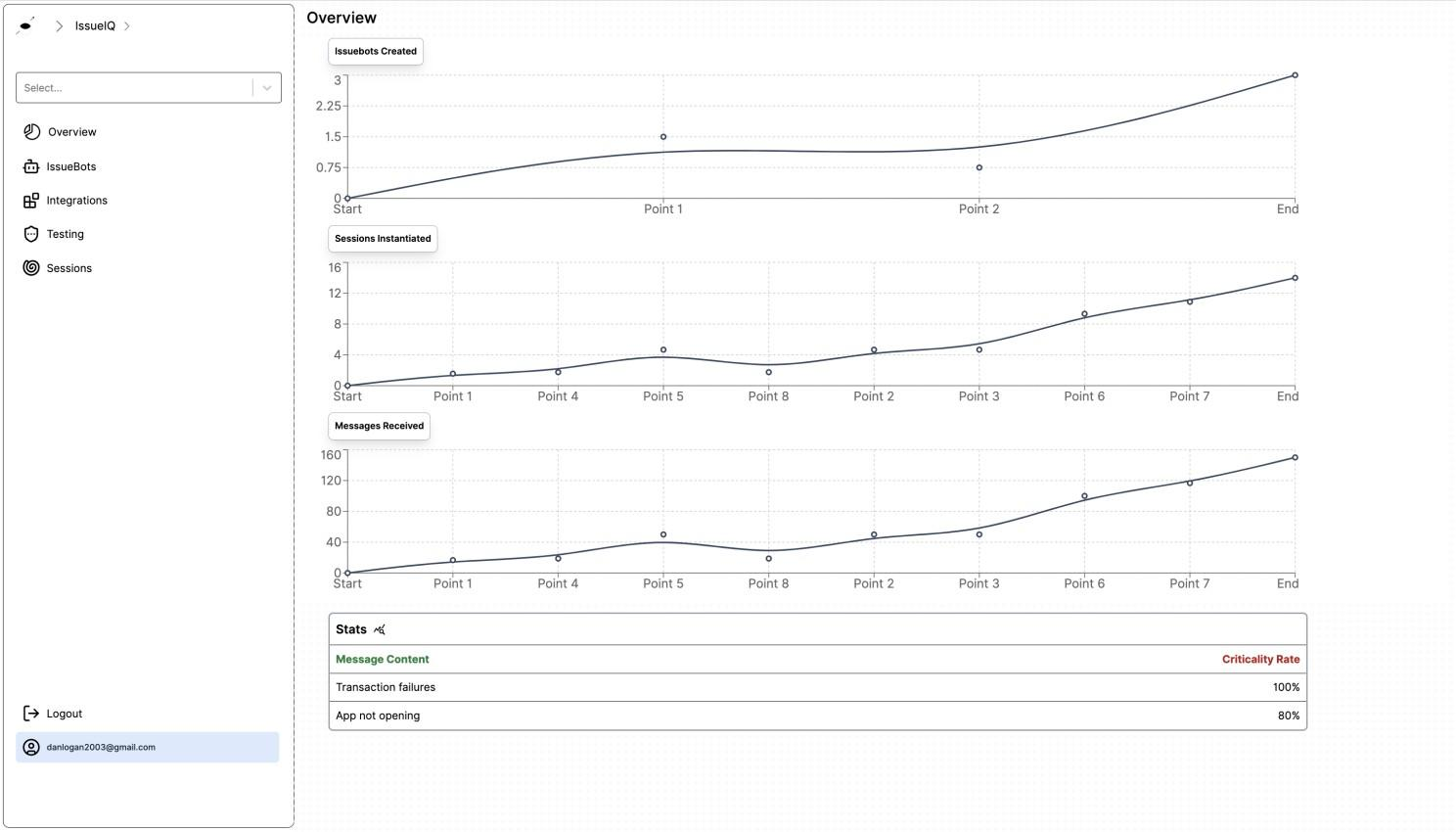
The Sessions Page displays a list of all active and past chatbot interactions. Users can monitor live conversations, review chat history, and analyze session details. This page helps users track customer interactions and improve chatbot responses based on real usage data. On this page, users can click the ‘Intercept’ button, which takes them to another page where they can directly interact with customers who are raising complaints.



**Figure 10.** Escalated sessions handling

**4.4.5 Analytics Dashboard**

The dashboard presents real-time and historical data about chatbot performance, user behavior, and escalation patterns. It also presents key performance metrics such as session count, average response time, escalation frequency, and resolution time. These metrics are retrieved from the PostgreSQL backend and visualized through interactive charting libraries. To ensure efficiency, background Celery tasks periodically aggregate raw logs into summary data, which is then cached and served to the frontend to minimize latency. This component offers system administrators actionable insights, enabling them to evaluate chatbot performance, adjust domain coverage, and optimize human agent allocation.



**Figure 11.** Overview and Analytics Page

**4.5 Deployment and Hosting**

The system was containerized using Docker and deployed to a cloud-based environment to enable scalability. PostgreSQL was hosted separately to handle persistent storage, while the WebSocket server and backend logic were deployed as independent microservices. The frontend was hosted on a static CDN for optimal load speed.

This implementation provides a production-ready foundation for businesses seeking to integrate intelligent, real-time customer support into their platforms. The next section evaluates the system’s performance and discusses its limitations and future directions.

**5. EVALUATION AND RESULTS**

This section evaluates the performance, usability, and reliability of the implemented real-time, embeddable RAG-based customer service system. The goal was to determine how well the system fulfilled its objectives in real-world or simulated scenarios, particularly in terms of response accuracy, escalation efficiency, and user satisfaction.

### ****5.1 Evaluation Strategy****

A hybrid evaluation approach was adopted, consisting of both **quantitative** and **qualitative** assessments. Key metrics included:

1. **Average Response Time:** Time between user input and bot/agent reply.
2. **Resolution Rate:** Percentage of sessions that concluded without requiring human intervention.
3. **Escalation Latency:** Time between escalation trigger and human agent response.
4. **Accuracy of Responses**: Measured by comparing bot answers to known intent file information.
5. **User Satisfaction**: Captured via feedback prompts after session completion.

The system was deployed in a controlled environment for testing, with simulated users engaging with the chatbot across various scenarios.

**5.2 Results**

The system demonstrated strong performance across key metrics during evaluation. Chatbot responses were consistently delivered in under three seconds, ensuring real-time interaction. Out of 150 test sessions, approximately 71% were fully resolved by the chatbot, while the remaining 29% required escalation to human agents.

Escalated cases were handled efficiently, with agents typically responding within 2.8 minutes of receiving a notification. The accuracy of chatbot responses was high, over 90%, when supported by well-structured intent files. However, performance dropped slightly with less organized or incomplete input data.

Overall, the results confirmed the system’s ability to deliver fast, accurate, and scalable customer service with effective human support when needed.

**5.3 Summary of Findings**

The evaluation demonstrated that the system:

1. Provides low-latency, accurate, and domain-aware chatbot responses,
2. Seamlessly escalates queries to human agents in real time,
3. Supports efficient administration through logging and analytics,
4. Is adaptable for deployment across industries requiring customer service automation.

The hybrid AI-human approach shows clear advantages over traditional standalone chatbot systems, especially in handling complex, multi-step queries where context and nuance are critical.

**6. CONCLUSION AND FUTURE WORK**

This study presented the design and implementation of a real-time, embeddable customer service system that integrates Retrieval-Augmented Generation (RAG) with human-in-the-loop (HITL) escalation. By combining large language model capabilities with contextual retrieval and agent intervention, the system offers a scalable and intelligent approach to modern customer support.

The system performed strongly during evaluation, demonstrating fast response times, high resolution rates, and seamless escalation processes. Users appreciated the intuitive interface, real-time interaction, and smooth transition between bot and human agents. Human agents also highlighted the value of session history access, which enabled them to respond with full conversational context. These features contributed to a user-friendly and effective hybrid support experience.

Nonetheless, some limitations emerged that point toward areas for future development. The system's performance was closely tied to the quality of the uploaded intent files; poorly structured or overly technical data reduced response accuracy. The chatbot also struggled occasionally with ambiguous or sarcastic input, leading to misinterpretations or unnecessary escalations. Additionally, the system currently supports only English, which limits its reach in multilingual environments.

Looking ahead, several enhancements are planned. Future iterations will explore:

1. **Multilingual support**, enabling broader deployment across diverse user bases;
2. **Advanced sentiment and tone detection**, improving the chatbot’s handling of emotionally nuanced queries;
3. **Voice-based interaction**, expanding accessibility;
4. **Adaptive learning**, allowing the system to improve over time based on user feedback and past interactions.

In conclusion, the system lays a strong foundation for intelligent customer support that blends AI automation with human empathy. With continued refinement, it holds promise as a flexible, enterprise-ready solution for organizations seeking to elevate their customer service capabilities.

Disclaimer (Artificial intelligence)

Option 2:

Author(s) hereby declare that generative AI technologies such as Large Language Models, etc. have been used during the writing or editing of manuscripts. This explanation will include the name, version, model, and source of the generative AI technology and as well as all input prompts provided to the generative AI technology

Details of the AI usage are given below:

1. The generation of Figure 1: Iterative Software Development Cycle (using ChatGPT with GPT-4o)

2. For editing some sections of the text (paraphrasing)

3.

**REFERENCES**

[1] M. Dixon, K. Freeman, and N. Toman, “Stop trying to delight your customers,” Harvard Business Review, vol. 88, no. 7/8, pp. 116–122, 2010.

[2] E. Adamopoulou and L. Moussiades, “Chatbots: History, technology, and applications,” Machine Learning with Applications, vol. 2, p. 100006, 2020.

[3] T. B. Brown et al., “Language models are few-shot learners,” in Advances in Neural Information Processing Systems, vol. 33, pp. 1877–1901, 2020.

[4] P. Lewis et al., “Retrieval-augmented generation for knowledge-intensive NLP tasks,” in Advances in Neural Information Processing Systems, vol. 33, pp. 9459–9474, 2020.

[5] Y. Xu, C.-H. Shieh, P. van Esch, and I.-L. Ling, “AI customer service: Task complexity, problem-solving ability, and usage intention,” Journal of Service Research, vol. 23, no. 4, pp. 458–473, 2020.

[6] G. Agrawal, S. Gummuluri, and C. Spera, “Beyond-RAG: Question identification and answer generation in real-time conversations,” arXiv preprint arXiv:2410.10136, Oct. 2024.

[7] A. Maurya and A. Bhattacharya, “Use of chatbots for customer service in MSMEs: Challenges and opportunities,” Applied Computing and Informatics, 2023.

[8] K. Pandya and M. Holia, “Automating customer service using LangChain: Building custom open-source GPT chatbot for organizations,” arXiv preprint arXiv:2310.05421, Oct. 2023.

[9] J. Weizenbaum, “ELIZA—a computer program for the study of natural language communication between man and machine,” Communications of the ACM, vol. 9, no. 1, pp. 36–45, 1966.

[10] B. AbuShawar and E. Atwell, “ALICE chatbot: Trials and outputs,” Computación y Sistemas, vol. 19, no. 4, pp. 625–632, 2015.

[11] T. B. Brown et al., “Language models are few-shot learners,” in Advances in Neural Information Processing Systems, vol. 33, pp. 1877–1901, 2020.

[12] I. Ullah, S. Ji, and C. Kim, “What (de)motivates customers to use AI-powered conversational agents: A behavioural reasoning approach,” Technological Forecasting and Social Change, vol. 188, p. 122318, 2023.

[13] A. Afzal, A. Kowsik, R. Fani, and F. Matthes, “Towards optimizing and evaluating a retrieval augmented QA chatbot using LLMs with human-in-the-loop,” in \*DaSH Workshop at NAACL\*, Mexico City, Apr. 2024.

[14] Q. Sun, T. Bi, S. Li, E.-J. Holden, P. Duuring, K. Niu, and W. Liu, “SymbioticRAG: Enhancing document intelligence through human‑LLM symbiotic collaboration,” arXiv preprint arXiv:2505.02418, May 2025.

[15] C. R. C. Lima, C. N. Carr, J. J. P. Margarido, and R. D. da Silva, “The incremental model in software development: a structured and interactive way to deliver quality products,” \*Research, Society and Development\*, vol. 12, no. 4, p. e7512440934, Mar. 2023.