ALISSAA AI

ANF Strategies

Developed for ISO New England

Manager: Michaela DeToma

Team: Abhignan Muppavaram, Andrew Corbett, Atharva Shahane, Ishita Abrol, Luke Lombard, Sameen Shaik, Srikiran Kavuri

Table of Contents

What is ALISSAA?

ALISSAA AI is a chatbot designed to assist in data retrieval from the ISO New England website. ALISSAA simplifies the processes of navigating the ISO New England site by answering technical and advanced questions without the user having to hunt down the information themselves. Relevant ISO New England questions can be input into the ALISSAA webpage and are responded to in a quick and accurate manner that ultimately saves time and resources.

PURPOSE & USERS:

This AI assists in retrieving relevant ISO New England data with ease, ensuring users can access information efficiently and accurately.

1. The majority of users of this software are expected to be experienced in the field of energy production and distribution: specifically, ISO employees in need of specific data or in customer service and distributing information to the inquiring clients.
2. Additionally, we expect General Public to find answers to general questions about ISO New England's structure, purpose, and policies in an intuitive and easy-to-understand format.

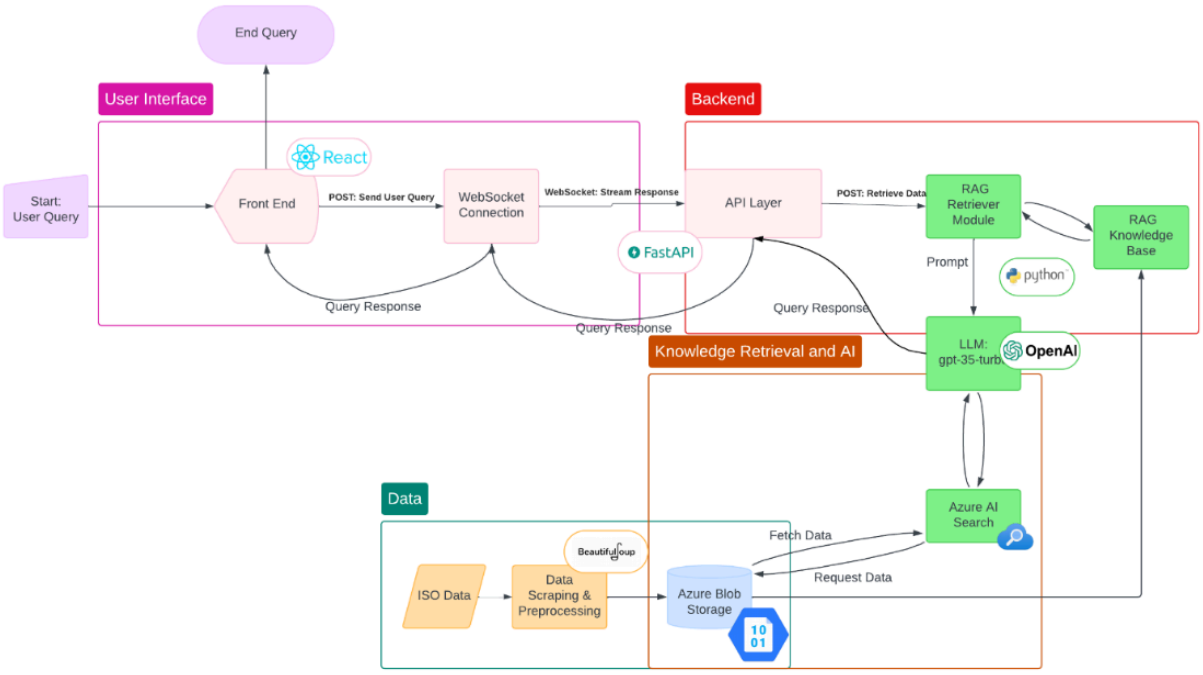
NON-FUNCTIONAL REQUIREMENTS:

1. Performance:
    - The chatbot will respond with minimum latency to improve user experience.
    - This will be achieved through Azure cloud services and caching mechanisms.
2. Security:
    - All user interactions and data integrations should follow strict security measures, including encrypted API keys and secure access to Azure Cognitive Search.
3. Accuracy:
    - Responses are cross-checked against a gold standard dataset to minimize AI-generated hallucinations and ensure that the answers are accurate.

Third Party Software

SOFTWARE TITLE	DESCRIPTION	PURPOSE
Azure AI Studio	AI development platform	LLM deployment and management, search and storage capabilities
React.js	JavaScript frontend framework	Building the user interface
Requests	Python library for making HTTP requests	Scraped PDF files from the ISO New England website
PyPDF2	PDF processing library	PDF document parsing and text extraction
Azure Blob Storage	Cloud storage service	Document storage and management
Azure Cognitive Search	Cloud-based search service	Index and search the scraped PDFs
OpenAI GPT-35-Turbo	Language Model	Core chatbot intelligence and response generation
Langchain	LLM framework	RAG (Retrieval Augmented Generation) pipeline implementation

Architectural Diagram



Encountered Difficulties

1. **Role Designation:** Initially, our team faced challenges with role distribution. The front-end development team was initially overstaffed, leading to inefficiencies in progress. To address this, we reorganized responsibilities, designating a single team member to handle the front end independently. This shift necessitated a steep learning curve for the assigned individual, as they had to self-learn React without prior experience. While this was a significant hurdle, it ultimately streamlined our workflow.

2. **Database Selection:** Selecting a suitable database for our chatbot was another major challenge. We deliberated between using blob storage and vector databases but struggled to find a solution compatible with our budget and requirements. Our team initially opted for Microsoft Azure, which offered completely free student credits. However, during the development phase, Azure revised its policies, requiring payment information for services we planned to use. This change disrupted our workflow and delayed progress across the stack.

   To resolve this, we calculated the estimated token usage for Azure's services and utilized a workaround working within the free $100 student credit limit by adding payment information that would never be used nor had risk in being used. This approach required collaboration across the team to refine our storage and processing strategy, ensuring cost efficiency while maintaining functionality.

3. **Integration and Time Constraints:** After resolving the storage issue, we shifted focus to integrating the front-end and back-end systems. However, the delays caused by earlier challenges significantly reduced the time available for this phase. To mitigate this, we appointed two leads to oversee and accelerate the integration process, ensuring tasks were prioritized and completed within the project timeline. This decision allowed us to complete the critical integration of our project.

What Worked Well

1. **Role Designation:** Our team maintained effective collaboration through regular meetings, productive discussions, and clear role assignments. Dividing into sub-teams (ex: front-end, back-end, database, etc.) allowed us to work efficiently and focus on specific tasks while ensuring alignment on project goals. This structure fostered a streamlined process that helped us meet milestones on time.

2. **Communication:** Within the team, we shared knowledge and supported each other in acquiring new skills. Additionally, we presented our progress to the client in a clear and professional manner, incorporating their feedback to align with client expectations.

3. **Adaptability:** Our team demonstrated resilience in addressing challenges, such as finding a workable database solution after encountering budget constraints with Azure. By collaborating on solutions and reallocating resources where needed, we adapted to setbacks and ensured progress across all project areas. This flexibility contributed to the successful integration of the front and back end under tight time constraints.

Technical Issues

Our first major technical challenge was with our initial LLM model, Ollama-Mistral. Although it showed promising results and produced accurate responses while using a small dataset (5 PDFs), the model hallucinated heavily on datasets that were only slightly larger (15 PDFs). This inaccuracy with the responses left us unsatisfied with this model and this is when we decided to shift to Microsoft Azure and utilize the LLM models provided by OpenAI.

Our team faced great difficulties using Microsoft Azure as this was the first time we had experience with this software. It took us a long time to get used to the features, roles, implementation, and execution of the model on Azure AI studio. Our first big hurdle was that we were unable to create a database using Azure Blob storage. We were able to solve this issue by working on the computer that made the hub on Azure, which was incredibly inconvenient and slowed down our pace.

After we were able to get the RAG model functioning, Azure decided to change its subscription policies, which was a huge hit to our team. The entire RAG environment we set up on Azure for 2 months was completely unusable because the guidelines and token quotas of the student subscription were changed, which means we weren't able to utilize our LLM at all.

On the bright side, we were able to solve this issue while switching to a pay-as-you-go subscription model (Student credits get used first, so our team is not spending any money), which granted us new quotas, making our RAG model usable again. Our team would like to thank Anthony from Team Fuze for helping us figure out this extremely critical issue.

Testing

To ensure the quality and reliability of our AI chatbot, we implemented a systematic testing approach centered on a "golden dataset." This dataset consisted of a curated collection of questions along with their ideal responses, which served as a benchmark for evaluating the chatbot's performance. Each team member contributed to the dataset by manually adding diverse and representative questions from various client-requested PDFs on the ISONewEngland website relevant to the chatbot's intended use cases.

During testing, the chatbot's responses were compared against the ideal answers in the dataset, and responses were assigned scores based on the accuracy and relevance of the outputs. This process allowed us to identify discrepancies, fine-tune the chatbot's underlying model, and iteratively improve its performance. By leveraging manual verification, we ensured our testing framework was both comprehensive and aligned with user expectations.

Using ALISSAA AI

To use ALISSAA AI, simply launch the application and input your query into the chat box provided on the user interface. ALISSAA is designed to quickly process your question, retrieve relevant information from ISO New England's indexed PDFs, and provide concise, accurate responses. Whether you are looking for operational data, technical procedures, or general policies, ALISSAA simplifies the process by eliminating the need to manually navigate complex documents. Users can enter any ISO-related query in natural language, and the chatbot will provide clear and informative answers, ensuring an intuitive experience.

To launch the chatbot, start the backend server and front-end application. First, install all dependencies necessary for the chatbot to run using the command pip install -r requirements.txt. Next ensure the backend is running by activating the virtual environment and executing the command uvicorn backend.main:app --reload. This will start the FastAPI server on http://127.0.0.1:8000. Then in a separate terminal, navigate to the React front-end directory using cd react-project and then run npm start. Once the front end launches, it will be accessible on http://localhost:3000. From there, you can interact with ALISSAA by typing your queries into the chatbox and receiving real-time responses powered by Azure's GPT-3.5-Turbo model and Azure Cognitive Search.