AIMSsist Chatbot System	
Final Project Proposal	
Course Code: CPE 201L	**Program:** BSCPE
Course Title: Data Structures and Algorithms	**Date Performed:**
Section: CpE-2A	**Date Submitted:**
Leader: Balaoro, Judge Wayne B. **Members:** Balana, Jerkielle Roen Barbas, Steven Jade Dispo, Lei Andrew T. Sorellano, John Kenneth	**Instructor:** Engr. Maria Rizette Sayo

1. Objective(s):

The main goal of this project is to design and implement a multi-user, persistent, queue-based chatbot application.

This is accomplished by two major objectives:

- The "AIMSsist" Chatbot would serve as a virtual assistant for the University of Caloocan City's AIMS Portal by answering students' most frequently asked questions about portal-related tasks such as enrollment, fee payment, checking grades, and downloading forms.

- The Queue data structure shall be applied to solve a real-world problem. The project shows the way one can organize incoming user requests with the help of a FIFO queue so that no data is lost and users may be served equitably in a multi-user environment.

2. Intended Learning Outcomes (ILOs):

- Understand how the Queue data structure works and its real-world application in system development.
- Apply FIFO (First In, First Out**)** logic to manage and process user requests from a shared, persistent queue.
- Gain experience in integrating data structures (like queues and lists) with a web application framework (`streamlit`) to enhance system functionality and user experience.
- Learn to record and retrieve user actions (messages) efficiently using algorithmic methods, specifically the `enqueue` and `dequeue` operations of a queue.
- Strengthen skills in Python programming (using classes, JSON for persistence, file I/O) and system design through the practical use of queues.

- Recognize the role of data structures in improving system organization and tracking, as seen in the `QueueManager` class which tracks all incoming, processing, and completed requests.

3. Discussion:

The AIMSsist Chatbot is a website designed to help students at the University of Caloocan City by answering their common questions about the AIMS portal. The Data Structure of this system is the QueueManager function, which acts as the organizer of the line. It has two jobs. When a student sends a message, the "enqueue" function adds that request to the back of the line. When the system is ready to answer a question, the "dequeue" function takes the request from the very front of the line. This guarantees that the first student to ask a question is always the first one to get an answer.

The biggest challenge was making sure the chatbot could handle a huge number of students asking questions at the same time without crashing, especially during busy periods like enrollment week. Our solution was to build the entire application around a "first-come, first-served" line, or a queue. This approach keeps the system stable and fair, ensuring that every student's question is answered in the order it was received and that no message is ever lost.

To make this line work for everyone using the website, we needed a way to save it in a javascript file. We use a simple shared file that acts as a central list of all the waiting questions. Every time a new question is asked, it is added to this list. The chatbot's engine constantly checks this list. Each time it checks, it takes the next question from the front, finds the answer, and then updates the list. By processing just one question at a time, the system can safely handle a massive flood of requests without ever becoming overwhelmed.

4. Materials and Equipments:

- Computer
- Streamlit
- Streamlit Community Cloud
- Python
- Pycharm
- Github

5. Procedure:

Step 1: Start the application (streamlit run dsa3.py).

Step 2: Initialize the main application loop (main function). Set up the page config, create a persistent AIMSKnowledgeBase (the "Answer Key"), and assign a unique user_id to the session.

Step 3: Load the shared queue: Call load_queue() to read the shared_queue_data.json file. If the file does not exist, create a new QueueManager and save it.

Step 4: Update the current user's activity by calling update_user_activity() and save the change by calling save_queue().

Step 5: Check for work: If the message_queue is not empty, call process_next_message() to process one item.

Step 6 (Dequeue Process): When process_next_message() is called:

- a. Call **dequeue()** to remove one item from the *front* of the message_queue.
- b. Call search_procedure() to find the matching answer from the AIMSKnowledgeBase.
- c. Add the finished answer to the *back* of the response_queue.
- d. Call save_queue() to save this new state (one less in message_queue, one more in response_queue).

Step 7: Display the full UI: Draw the chat window, system status, and the Queue Monitor (showing message_queue length and active users).

Step 8: Check for pickup: Call get_response() to search the response_queue for an answer matching the current user_id. If found, remove it, add it to the user's personal chat, and call save_queue().

Step 9 (Enqueue Process): When a user sends a message (either by typing or clicking a button), call the process_user_question() function.

Step 10: When process_user_question() is called:

- a. Call load_queue() to get the most recent line.
- b. Call add_user_message(), which in turn calls **enqueue()** to add the new message to the *back* of the message_queue.
- c. Call save_queue() to immediately save this new addition to the file.

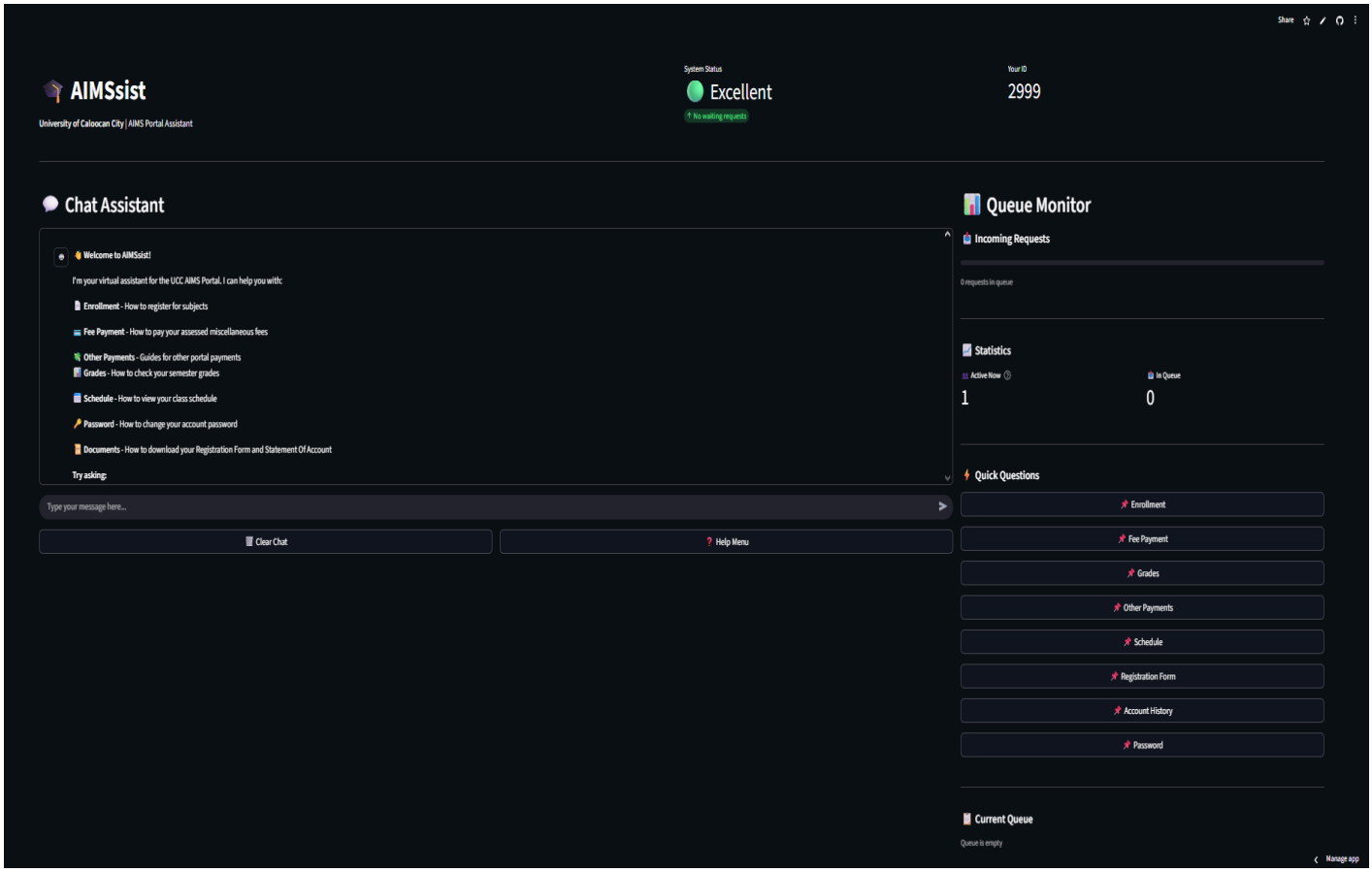
Step 11: Before any operation, check queue status using get_queue_health() to display the system status.

Step 12: Display the message_queue (up to 5 items) in the Queue Monitor so the user can see the live line.

Step 13: Check auto-refresh: If the user is waiting, the system is processing, OR the message_queue is not empty, wait 0.5 seconds and call st.rerun() to go back to **Step 3**.

Step 14: End (If no refresh is triggered, the app becomes idle and waits for user input, which restarts the loop at Step 9).

6. Output



7. Conclusion:

In conclusion, our application acts as a well-organized model of a queue-oriented chatbot system known as AIMSsist, Using Streamlit. It efficiently manages several users by utilizing a message queuing system via the QueueManager class, ensuring that user messages are handled systematically and equitably. The AIMSKnowledgeBase class delivers responses based on keywords, allowing the system to handle frequent questions or show a standard help menu if no matching result is identified. The incorporation of a common JSON file enables data retention across sessions, ensuring message continuity and ongoing user monitoring. The Streamlit interface improves user experience by offering an interactive chat setting alongside current queue data and system performance metrics.

Although the program showcases a solid structure with distinct modular design and efficient management of simultaneous users, it is constrained by its single-threaded execution, keyword reliance, and dependence on local file storage. In general, the application effectively demonstrates the fundamental concepts of a multi-user conversational queue system and offers a solid foundation for future improvements like real-time asynchronous processing, enhanced natural language comprehension, and database integration. Even though we face many hardships while doing this program such as having a problem with the payment method in enrolling we manage to do our best as a team and by that we believe that the program we made is a success.

8. References

Purnama, E. B., Azizah, N., Nuraminudin, M., & Mustika Dewi, M. *(n.d.). Analysis and design of queue service information system integrated with WhatsApp using UML method. International Journal of Computer and Information System (IJCIS). Retrieved from* *https://www.ijcis.net/index.php/ijcis/article/view/151*

Qaffas, A. A. *(2019). Improvement of chatbots semantics using Wit.ai and word sequence kernel: Education chatbot as a case study. International Journal of Modern Education and Computer Science (IJMECS), 11(3), 16–22. https://doi.org/10.5815/ijmecs.2019.03.03*

Jia, J. *(2003). The study of the application of a keywords-based chatbot system on the teaching of foreign languages. arXiv preprint. Retrieved from https://arxiv.org/abs/cs/0310018*

Altinok, D. *(2018). An ontology-based dialogue management system for banking and finance dialogue systems. arXiv preprint. Retrieved from https://arxiv.org/abs/1804.04838*

Saxena, A. *(2022). AI-based chatbot. Journal for Research in Applied Science and Engineering Technology (IJRASET), 10(1), 47785. https://doi.org/10.22214/ijraset.2022.47785*

Al-Matar, N. *(2021). Analysis of some communication-related queuing system modeling and performance. Journal of System Engineering and Technological Innovation, 2(2), 51–58. https://doi.org/10.38156/jisti.v2i02.51*

Uddin, M. N., Uddin, M. N., & Uddin, M. N. *(2016). Automated queue management system. Global Journal of Management and Business Research, 16(A1), 51–58. Retrieved from https://journalofbusiness.org/index.php/GJMBR/article/view/1898*