A Mini-Project Report

on

**INTELLIGENT COLLEGE ASSISTANT**

Submitted for partial fulfillment of the requirements for the award of the degree

of

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**BY**

**Mr. Thimirishetty Karthikeya (2451-20-733-152)**

**Mr. Mohammed Rafae Ahmed (2451-20-733-162)**

**Mr. Surya Danturty (2451-20-733-163)**

Under the guidance of

**Mrs. Bantu Saritha**

(Associate Professor CSE dept)

****

Department of Computer Science and Engineering

Maturi Venkata Subba Rao Engineering College

(An Autonomous Institution)

(Affiliated to Osmania University & Recognized by AICTE)

Nadergul(V), Balapur(M), RR Dist. Hyderabad – 501 510

2022-23.

**Maturi Venkata Subba Rao Engineering College**

**(An Autonomous Institution)**

(Affiliated to Osmania University, Hyderabad)

Nadergul(V), Hyderabad-501510



**Certificate**

This is to certify that the mini-project work entitled “**INTELLIGENT COLLEGE ASSISTANT”** is a bonafide work carried out by **Mr. Thimirishetty Karthikeya (2451-20-733-152), Mr. Mohammed Rafae Ahmed (2451-20-733-162), Mr. Surya Danturty (2451-20-733-163)** in partial fulfillment of the requirements for the award of degree of **BACHELOR OF ENGINEERING IN COMPUTER SCIENCE AND ENGINEERING** from Maturi Venkata Subba Rao Engineering College, affiliated to OSMANIA UNIVERSITY, Hyderabad, under our guidance and supervision.

The results embodied in this report have not been submitted to any other university or institute for the award of any degree or diploma.

**Internal Guide Project Co-Ordinators**

**Mrs. Bantu Saritha**

Associate Professor

**DECLARATION**

This is to certify that the work reported in the present mini-project entitled “**INTELLIGENT COLLEGE ASSISTANT”**  is a record of bonafide work done by us in the Department of Computer Science and Engineering, Maturi Venkata Subba Rao Engineering College, Osmania University. The reports are based on the mini-project work done entirely by us and not copied from any other source.

The results embodied in this mini-project report have not been submitted to any other University or Institute for the award of any degree or diploma to the best of our/ my knowledge and belief.

Thimirishetty Karthikeya Mohammed Rafae Ahmed Surya Danturty

(2451-20-733-152) (2451-20-733-162) (2451-20-733-163)

**ACKNOWLEDGEMENTS**

We would like to express our sincere gratitude and indebtedness to my mini-project guide **Mrs. Saritha Bantu** for her valuable suggestions and interest throughout the course of this mini-project.

We are also thankful to our principal **Dr.** **G. Kanakadurga** and **Mr. J Prasanna Kumar**, Professor and Head, Department of Computer Science and Engineering, Maturi Venkata Subba Rao Engineering College, Hyderabad for providing excellent infrastructure for completing this mini-project successfully as a part of our B.E. Degree (CSE). We would like to thank our mini-project coordinator(s) **Mr.K Murali Krishna,Mrs. N Sabitha, Dr. Namita Parati** for their constant monitoring, guidance, and support.

We convey our heartfelt thanks to the lab staff for allowing me to use the required equipment whenever needed.

Finally, we would like to take this opportunity to thank my family for their support through the work. We sincerely acknowledge and thank all those who gave directly or indirectly their support in the completion of this work.

Thimirishetty Karthikeya (2451-20-733-152)

Mohammed Rafae Ahmed (2451-20-733-162)

Surya Danturty (2451-20-733-163)

**ABSTRACT**

The project aims to develop an intelligent chatbot system integrated with the college website using OpenAI API and advanced technologies. The existing college website lacks an efficient communication channel, requiring users to navigate through various pages to find information. The proposed chatbot system addresses this limitation by providing a user-friendly interface for students, faculty, and visitors to interact with the website. Leveraging natural language processing and machine learning algorithms, the chatbot can understand user queries and provide accurate and personalized responses related to courses, faculty members, and other important details. By utilizing LangChain Vector Database and OpenAI API, the chatbot enhances the user experience, streamlines information retrieval, and offers quick access to relevant information, ultimately improving the overall efficiency of the college website.

**TABLE OF CONTENTS**

PAGE NOS.

Certificate ...................................................................................................... i

Declaration .................................................................................................... ii

Acknowledgments......................................................................................... iii

Abstract.......................................................................................................... iv

Table of contents............................................................................................. v-vi

List of Figures................................................................................................. vii

**CONTENTS**

CHAPTER I

1. INTRODUCTION 01 - 02
   1. PROBLEM STATEMENT 01
   2. EXISTING SYSTEM 01
   3. PROPOSED SYSTEM 01
   4. SCOPE OF THE MINI-PROJECT 02

CHAPTER II

1. TOOLS AND TECHNOLOGIES 03 - 05
   1. LITERATURE SURVEY 03
   2. HARDWARE REQUIREMENTS 04
   3. SOFTWARE REQUIREMENTS 04 - 05

CHAPTER III

1. SYSTEM DESIGN 06 - 12
   1. SYSTEM ARCHITECTURE 06 - 08
   2. UML DIAGRAMS 09 - 12

CHAPTER IV

1. SYSTEM IMPLEMENTATION & METHODOLOGIES 13 - 18
   1. DEVELOPMENT ENVIRONMENT SETUP 13 - 14
   2. IMPLEMENTATION DETAILS 14 - 18
   3. ALGORITHM 18 - 19

CHAPTER V

1. TESTING / RESULTS 20 - 23
   1. TEST CASES (SCREENSHOTS) 20 - 23

CHAPTER VI

1. CONCLUSION & FUTURE ENHANCEMENTS 24 - 25

REFERENCES/ BIBLIOGRAPHY 26

APPENDIX 27 - 31

|  |
| --- |
| **LIST OF FIGURES**  **Figure no Figure Name Page no**  **Figure 3.1 System Architecture 8**  **Figure 3.2 User Interactions and System Functionalities 9**  **Figure 3.3** **Structural Overview of Classes and Relationships 10**  **Figure 3.4 Object States and Transitions 11**  **Figure 3.5 Flow of Activities and Decision Points 11**  **Figure 3.6 Component Interactions and Message Flows 12**  **Figure 4.1 User Interface 15**  **Figure 4.2 Flow of Code 15**  **Figure 5.1 Achievements Q&A 20**  **Figure 5.2 Publications Q&A 21**  **Figure 5.3 About faculty Q&A 22**  **Figure 5.4 Chatbot response to invalid input 22**  **Figure 5.5 Chatbot response to out-of-scope question 23** |

**CHAPTER 1**

**INTRODUCTION**

* 1. **Problem Statement:**

The current college website lacks an efficient and interactive communication channel to cater to the needs and queries of the students, faculty, and visitors. The traditional website structure makes it difficult for users to quickly access relevant information and receive personalized assistance. There is a growing need for a chatbot solution that utilizes the latest technologies, such as LangChain Vector Database and OpenAI APIs, to enhance the user experience and provide accurate and personalized information.

The goal of this project is to design and develop an intelligent chatbot that integrates with the college website and acts as a virtual assistant for students, faculty, and visitors.

**1.2 Existing System:**

The existing college website lacks an efficient communication channel for users to quickly access relevant information and obtain personalized assistance. Users are required to navigate through the entire website, which can be time-consuming and frustrating. There is a pressing need for a chatbot solution that integrates seamlessly with the college website to enhance the user experience and provide instant, accurate, and personalized information.

**1.3 Proposed System:**

To address the limitations of the existing college website, we propose the development of an intelligent chatbot that seamlessly integrates with the website, revolutionizing the way users access information and receive assistance. The chatbot will serve as a virtual assistant, providing instant and personalized responses to user queries.

The objective of this project is to develop an intelligent chatbot that acts as a reliable and efficient communication channel for users of the college website. The chatbot will significantly enhance the user experience by providing instant access to personalized information and assistance, eliminating the need for users to navigate through the entire website. The implementation of this chatbot will revolutionize the way students, faculty, and visitors interact with the college website, ultimately improving overall satisfaction and engagement.

**1.4 Scope of Mini-Project:**

The primary focus of the chatbot is to enhance the user experience by providing quick access to information and personalized assistance. The following aspects are within the scope of the project:

* **Chatbot Features**: The project will encompass the development of essential chatbot features, such as natural language processing, information retrieval, personalized assistance, contextual understanding, multilingual support, and continuous learning.
* **Database and Content Integration:** The chatbot will retrieve and utilize information from the college website's database, including course catalogs, faculty directories, admission guidelines, and other relevant data. It will leverage this information to provide accurate responses to user queries.
* **User Interface Design:** The project will include the design and development of an interactive and user-friendly chat interface for users to engage with the chatbot. The interface will facilitate natural language conversations, offer prompts and suggestions, and provide a seamless conversational experience.
* **System Design**: Designing the architecture and user interface of the chatbot system, considering factors such as user experience, scalability, and integration with the existing college website. Creating design specifications and user flow diagrams to guide the development process.

**CHAPTER 2**

**TOOLS AND TECHNOLOGIES**

This chapter focuses on the tools and technologies relevant to our project of developing an intelligent college assistant chatbot. Before delving into the details, let's provide a brief introduction to the chapter. The rise of ChatGPT and virtual assistants has led to significant advancements in improving user experiences and information retrieval. In this context, we conducted a literature survey to explore existing research and advancements in chatbot development.

The literature survey revealed three main types of chatbot technologies: rule-based systems, retrieval-based models, and generative models. Additionally, we explored various natural language processing (NLP) techniques such as named entity recognition, sentiment analysis, and intent classification, which enhance chatbot language understanding. Knowledge representation and retrieval techniques, including knowledge bases, graph databases, and semantic search, were also examined. Based on the survey findings, we proposed the development of an intelligent college assistant chatbot that utilizes OpenAI APIs and Langchain technologies to bridge the gap between information retrieval and user accessibility, facilitating a user-friendly interface for students to navigate their academic journey efficiently.

In the next section, we will outline the hardware and software requirements necessary for the development and implementation of our intelligent college assistant chatbot.

**2.1 LITERATURE SURVEY**:

The rise in the development of ChatGPT and virtual assistants has gained significant attention in recent years, with a focus on improving user experiences and information retrieval. In the context of our project, which aims to create an intelligent college assistant chatbot, we conducted a literature survey to explore the existing research and advancements in this field.

**Chatbot Technologies:**

* Rule-based systems: Predefined rules and patterns for generating responses.
* Retrieval-based models: Predefined responses based on similarity matching.
* Generative models: Deep learning-based responses generated from training data.

**Natural Language Processing (NLP) Techniques:**

* Named entity recognition, sentiment analysis, and intent classification enhance chatbot language understanding.
* Word embeddings and language models, like OpenAI's GPT, improve the quality of chatbot interactions

**Knowledge Representation and Retrieval:**

* Knowledge bases, graph databases, and vector stores store and retrieve information for chatbots.
* Techniques like entity linking and semantic search improve retrieval processes and ensure accurate answers.

Based on our literature survey, we have identified the gaps in the existing approaches and proposed the development of an intelligent college assistant chatbot. By leveraging OpenAI APIs and Langchain technologies, we aim to bridge the gap between information retrieval from the college website and user accessibility. Our project seeks to provide a user-friendly interface where students can easily obtain information related to the college and navigate their academic journey more efficiently.

**2.2 HARDWARE REQUIREMENTS:**

* Processor: Intel Core i5 or higher
* RAM: 8 GB or higher
* Internet Connection: Stable internet connection for accessing APIs and online resources

**2.3 SOFTWARE REQUIREMENTS:**

* Operating System: Windows, macOS, or Linux
* Python: Version 3.7 or higher
* Integrated Development Environment (IDE): Visual Studio Code or any preferred Python IDE
* Dependencies:
* streamlit
* python-dotenv
* PyPDF2
* langchain
* numpy
* faiss-cpu
* OpenAI API

These software requirements provide the necessary infrastructure for developing and deploying the chatbot system.

**CHAPTER 3**

**SYSTEM DESIGN**

This section delves into the system design of the intelligent college assistant chatbot. It provides an overview of the system architecture and UML diagrams, which capture the design and interactions of the chatbot system. The system architecture showcases the components, modules, and their relationships, emphasizing the chatbot's integration with the college website and its role in improving communication channels. The UML diagrams present visual representations, including use case diagrams, class diagrams, sequence diagrams, activity diagrams, and state machine diagrams. These diagrams illustrate the chatbot's functionalities, interactions, and underlying design patterns. Understanding the system design and UML diagrams will give insights into the structure and efficiency of the chatbot system.

**3.1 SYSTEM ARCHITECTURE:**

The system architecture for our intelligent chatbot system integrated with the college website is designed to enhance communication and information retrieval for students, faculty, and visitors. The architecture consists of several key components working together seamlessly.

At the front end, we have a user-friendly interface provided by the Streamlit framework. This interface allows users to interact with the chatbot by inputting questions or queries related to the college and its information.

The system leverages natural language processing (NLP) techniques to understand user intent and extract relevant information from their input. This is achieved by utilizing the OpenAI API, which provides advanced language modeling and question-answering capabilities.

To provide comprehensive information, we extract text data from PDF documents that contain faculty data, achievements, publications, and membership details. We utilize the PyPDF2 library to read the PDF files and extract text from each page.

To optimize processing and improve efficiency, the extracted text is then split into smaller chunks using the CharacterTextSplitter from the LangChain module. These chunks allow for better management and faster retrieval of information.

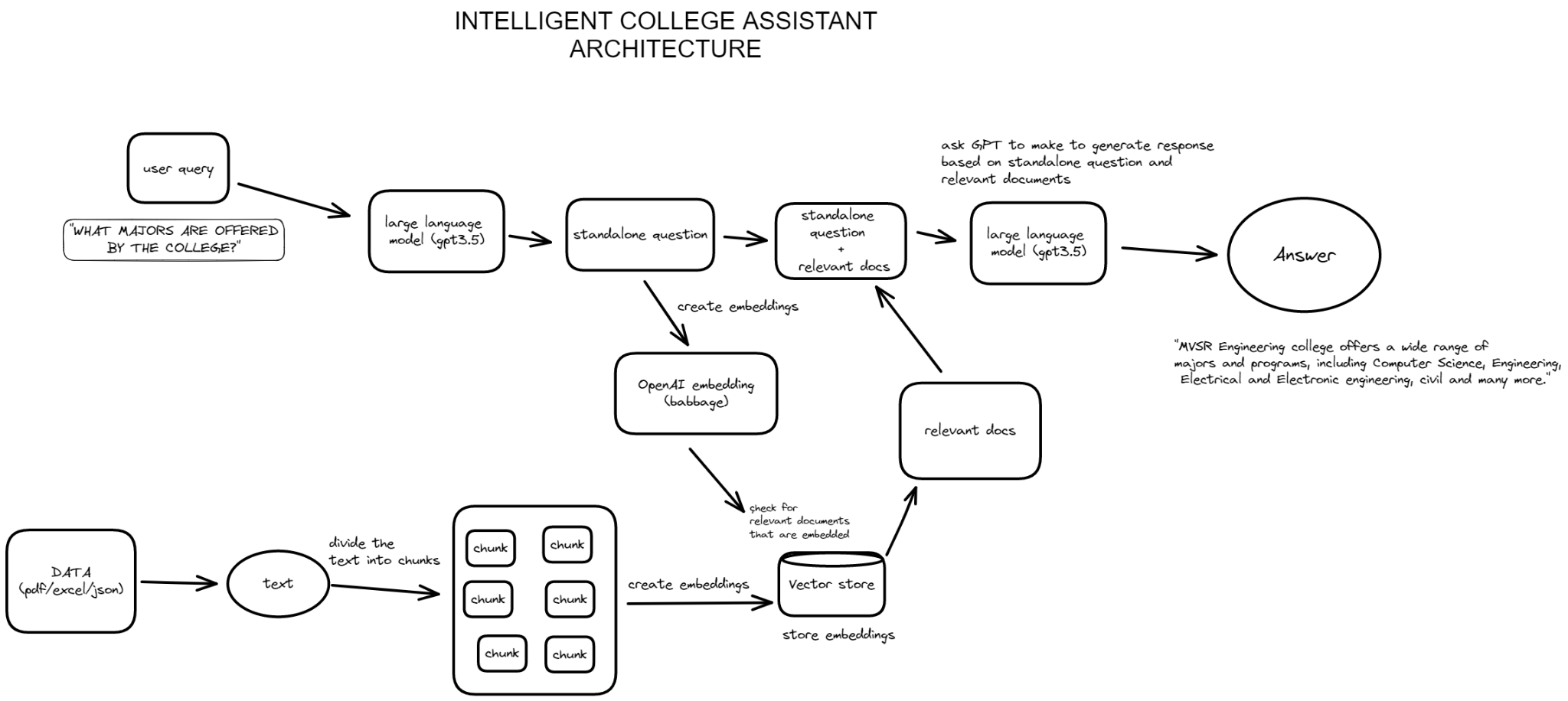
The system creates a vector store using the OpenAIEmbeddings module and the FAISS (Facebook AI Similarity Search) module. The OpenAIEmbeddings module generates embeddings for each text chunk, while the FAISS module creates a vector store that efficiently indexes and retrieves similar texts.

To enable interactive conversations, a conversation chain is implemented using the ChatOpenAI class from the LangChain module. This chain integrates the language model, the vector store, and a conversation buffer memory. The conversation buffer memory stores the chat history and enables seamless back-and-forth interactions between users and the chatbot.

When a user enters a question, it is passed to the conversation chain for processing. The chatbot generates a response by leveraging the language model, retrieving relevant information from the vector store, and considering the chat history. The response is then displayed to the user through the user interface, providing accurate and personalized information.

The system also allows for user feedback, which can be provided on the chatbot's responses. This feedback is stored in the conversation buffer memory, enabling continuous improvement of the system over time.

Overall, our intelligent chatbot system with its carefully designed architecture enhances communication, streamlines information retrieval, and improves the user experience on the college website.

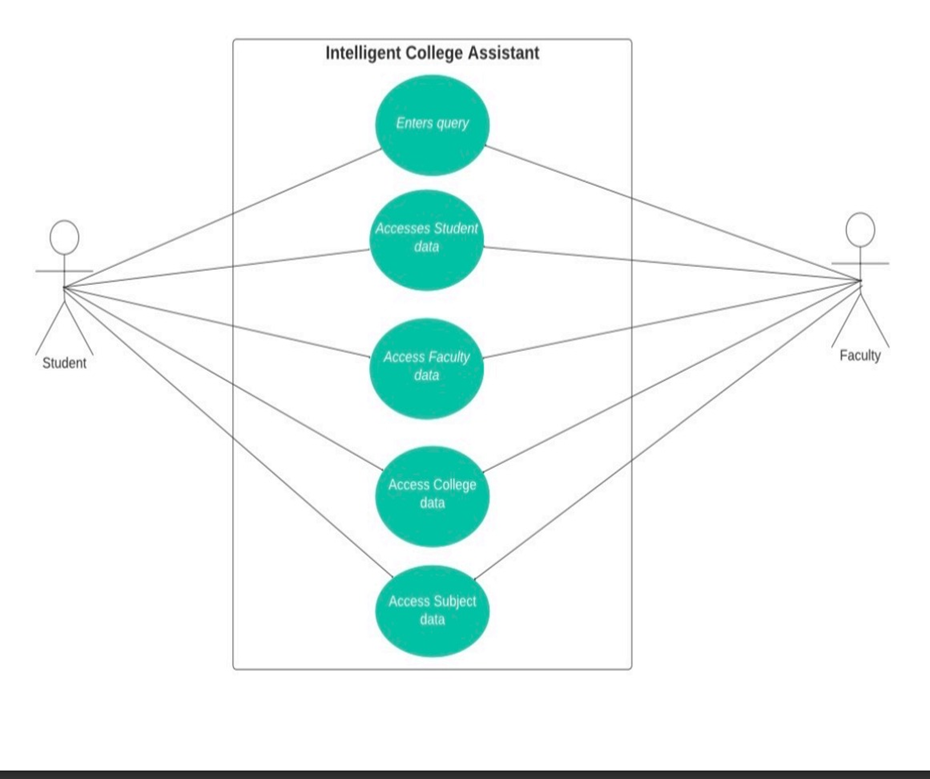


*Figure 3.1 System Archiecture*

**3.2 UML DIAGRAMS:**

**Use case diagram:**

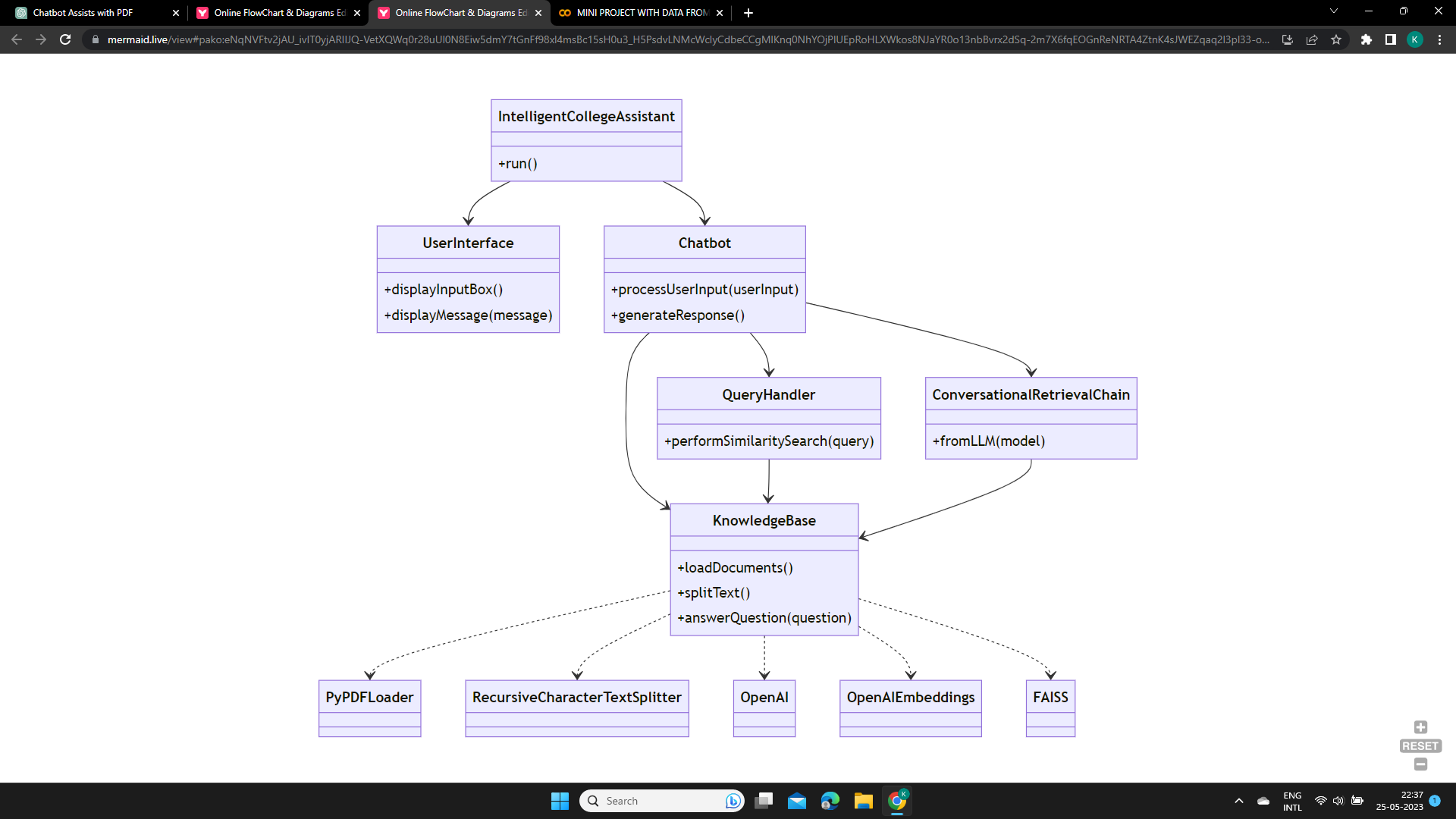
The figure 3.2 illustrates the main functionalities and interactions between users and the chatbot system. It helps identify the different user roles and the tasks they can perform within the system.

****

*Figure 3.2 User Interactions and System Functionalities*

**Class diagram:**

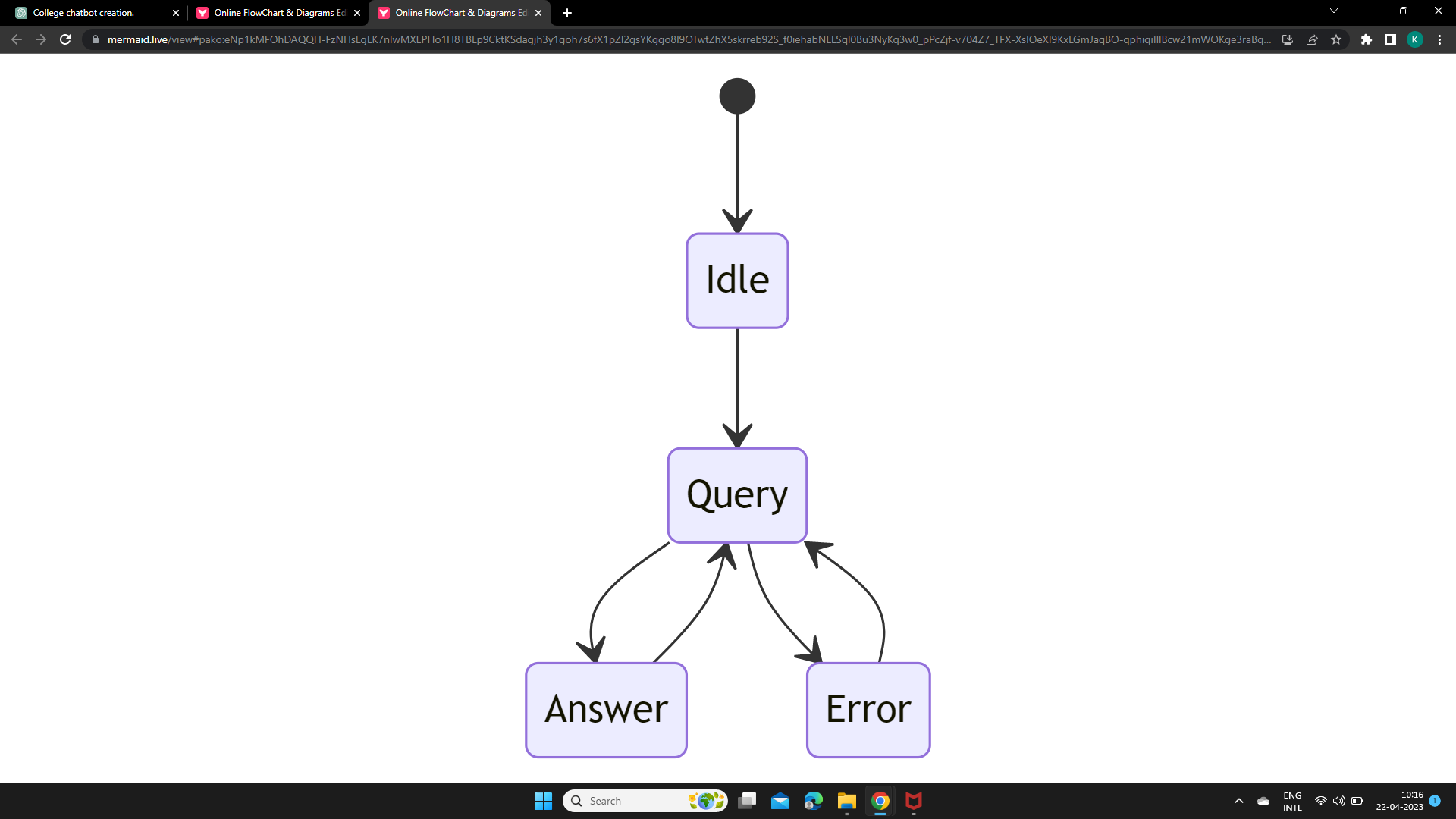
The figure 3.3 Shows the structure and relationships between classes in the chatbot system. It provides an overview of the system's object-oriented design, showcasing the entities and their associations.

****

*Figure 3.3 Structural Overview of Classes and Relationships*

**State diagram:**

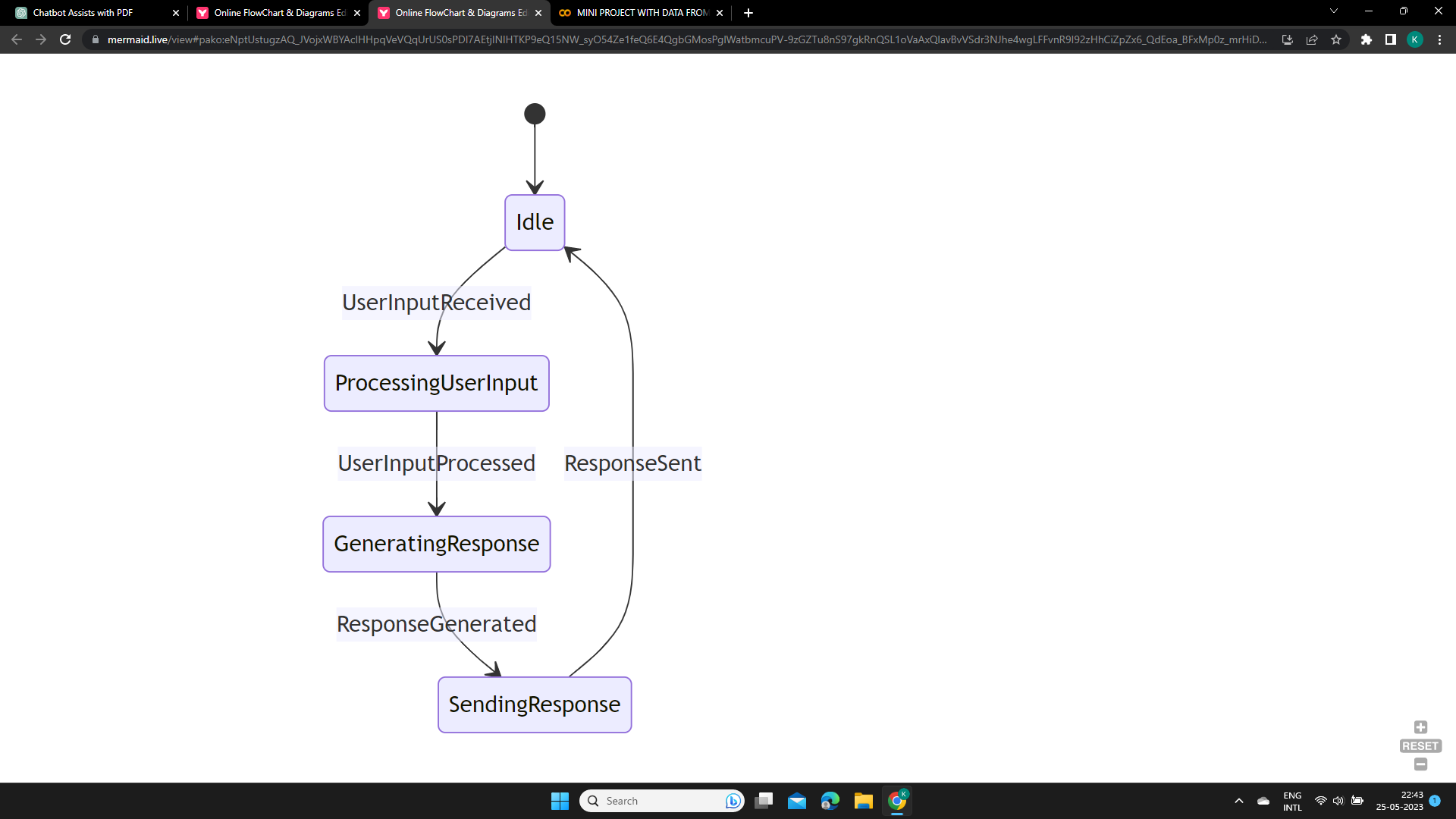
Figure 3.4 Models the different states and transitions of an object or system within the chatbot system. It illustrates the behavior and changes in states based on events or conditions

****

*Figure 3.4 Object States and Transitions*

**Activity diagram:**

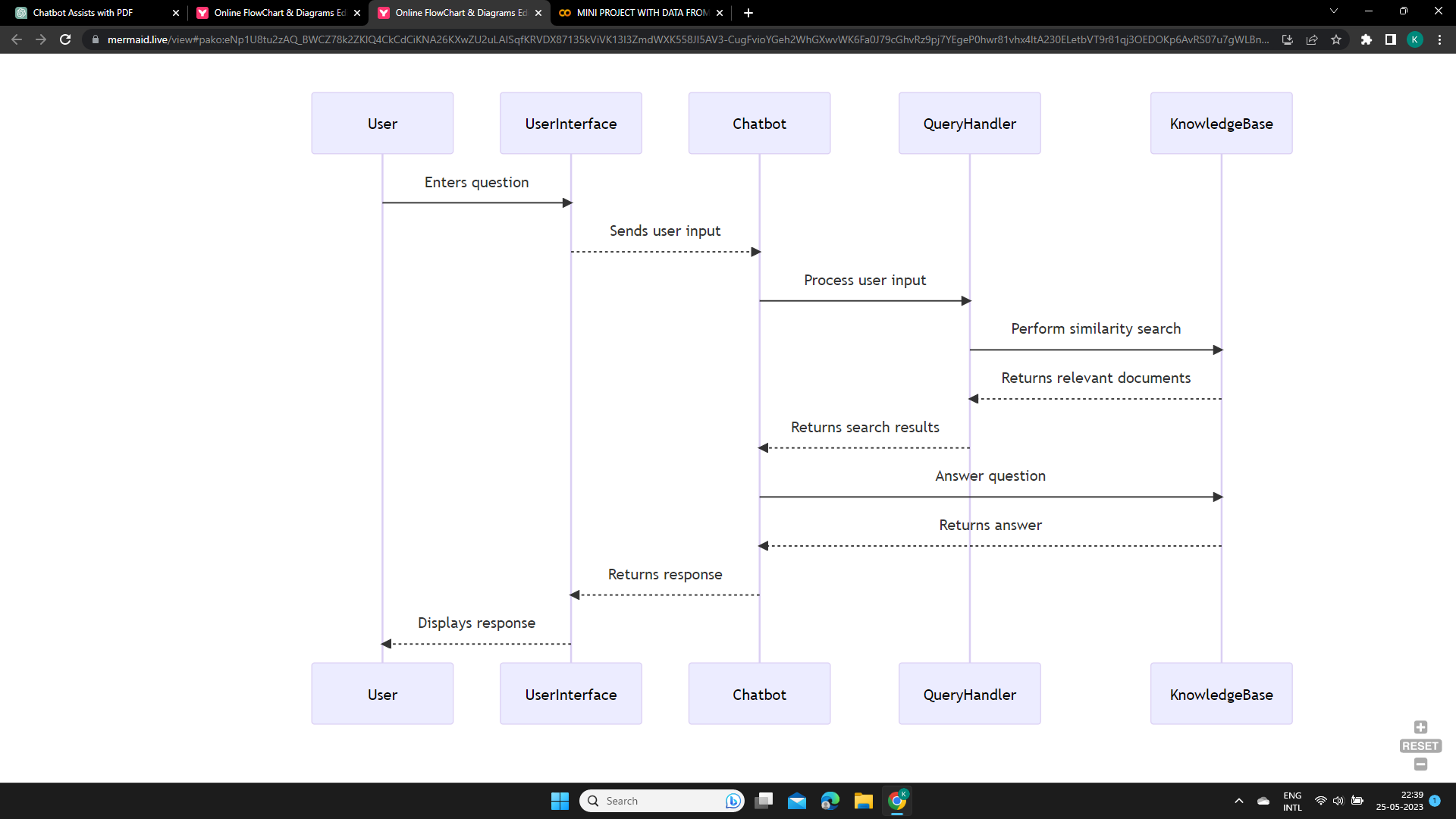
The figure 3.5 represents the flow of activities or processes within the chatbot system. It shows the steps involved in accomplishing a specific task or use case, including decision points and parallel flows.

****

*Figure 3.5 Flow of Activities and Decision Points*

**Sequence diagram:**

The figure 3.6 depicts the sequence of interactions between components or objects in the chatbot system for a specific use case. It illustrates the flow of messages and the order in which actions are executed.

* Figure 3.6 Component Interactions and Message Flows*

**CHAPTER 4**

**SYSTEM IMPLEMENTATION & METHODOLOGIES**

In this chapter, we will cover the setup of the development environment for the intelligent college assistant chatbot system. We will walk through the steps to install Python, required packages, and configure API settings. Additionally, we will discuss the implementation details, including the user interface, PDF text extraction, text chunking, vector store creation, conversation chain, handling user input, displaying chat history, and the main function's role. Lastly, we will provide a sample code snippet to illustrate the implementation.

**4.1 Development Environment Setup**

To set up the development environment for the intelligent college assistant chatbot system, follow these steps:

1. **Install Python**:

- Download the latest version of Python from the official Python website.

- Run the installer and follow the on-screen instructions to complete the installation.

- Make sure to select the option to add Python to the system PATH during the installation process.

2. **Install Required Packages**:

- Open a command prompt or terminal.

- Navigate to the project directory using the `cd` command.

- Run the following command to install the necessary packages:

```

pip install streamlit dotenv PyPDF2 langchain

```

This will install the required packages for running the chatbot system.

3**. API Configurations**:

- Obtain an OpenAI API key by signing up for an account on the OpenAI website.

- Create a file named `.env` in the project directory.

- Add the following line to the `.env` file, replacing `YOUR\_API\_KEY` with your actual OpenAI API key:

```

OPENAI\_API\_KEY=YOUR\_API\_KEY

```

- Save the `.env` file.

4. **Environment Setup Verification:**

- Run the following command to verify that the development environment is set up correctly:

```

python app.py

```

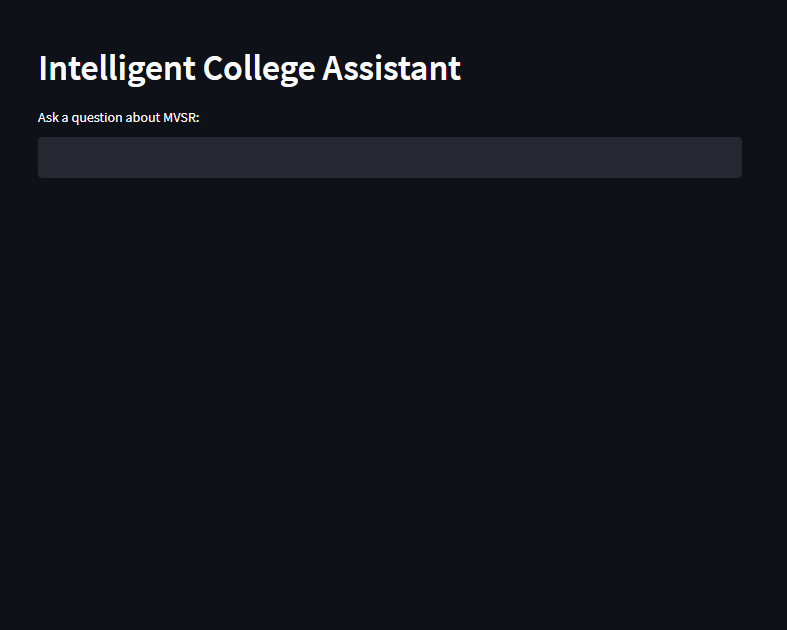
- If the development environment is properly configured, the chatbot system will start running and can be accessed through the provided URL.

By following these steps, we can successfully set up the development environment for the intelligent college assistant chatbot system.

**4.2 IMPLEMENTATION DETAILS:**

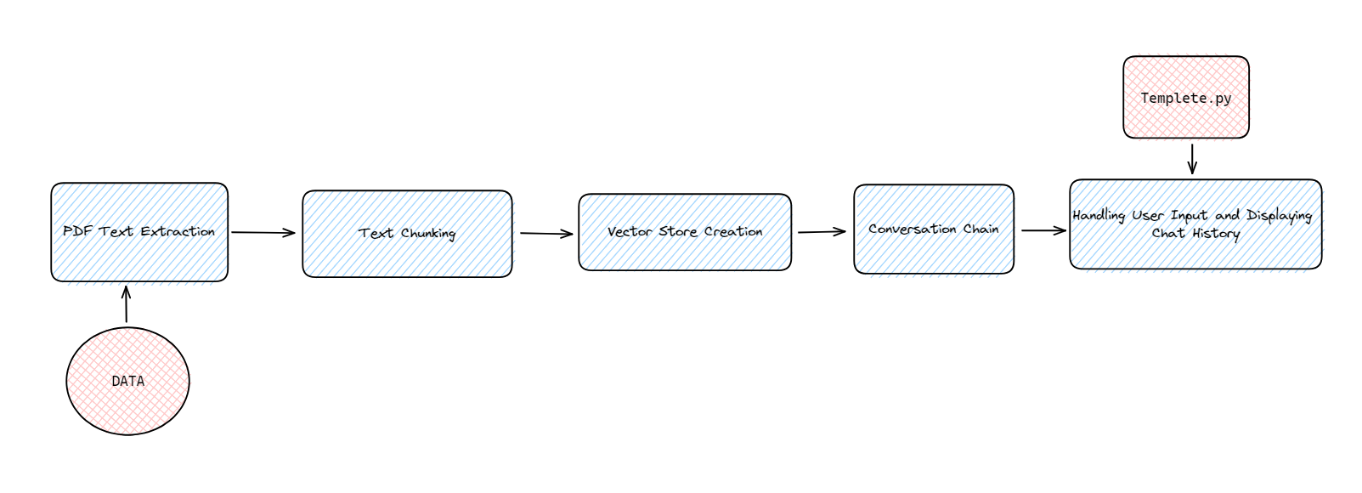
**User Interface:**

* As shown in the figure 4.1 the user interface is implemented using the Streamlit framework, which provides a user-friendly interface for users to interact with the chatbot.
* The st.set\_page\_config function was utilized to configure the page title and icon for the interface.
* The ‘template’ HTML code was used to define the styling of the chat messages.
* The st.header and st.text\_input functions were employed to create the header and input field for user questions.



*Figure 4.1 User Interface*

Before delving into the implementation details, it is helpful to understand the high-level flow of the code. The following flowchart provides an overview of how the code progresses through various stages and functionalities. This flowchart illustrates the key steps involved in the execution of the intelligent college assistant chatbot system.



*FIGURE 4.2 Flow of Code*

**PDF Text Extraction:**

* The PyPDF2 library is used to extract text data from PDF documents.
* The get\_pdf\_text function takes a list of PDF documents as input and uses the PdfReader class from PyPDF2 to read the PDF files.
* The extracted text from each page of the PDF documents is concatenated and returned as a single string.

**Text Chunking:**

* The get\_text\_chunks function splits the extracted text into smaller chunks to optimize processing.
* The CharacterTextSplitter class from the LangChain module is used to split the text based on specified parameters such as chunk size, overlap, and length function.
* The text is split into manageable chunks of a specified length, allowing for more efficient retrieval and processing.

**Vector Store Creation:**

* The get\_vectorstore function utilizes the OpenAIEmbeddings class from the LangChain module to generate embeddings for each text chunk.
* These embeddings capture the semantic meaning of the text and provide a representation for similarity search.
* The FAISS (Facebook AI Similarity Search) module is employed to create a vector store from the text chunks.
* The FAISS.from\_texts function is used to create the vector store, which efficiently indexes the text chunks for fast retrieval based on similarity.

**Conversation Chain:**

* The conversation chain is implemented using the ChatOpenAI class from the LangChain module.
* The get\_conversation\_chain function sets up the conversation chain by integrating the language model (llm), the vector store converted into a retriever, and a conversation buffer memory.
* The conversation buffer memory, defined as memory, stores the chat history and allows for interactive conversations between the user and the chatbot.
* The conversation chain utilizes the language model and the vector store to generate responses to user queries and considers the chat history to maintain context during the conversation.

**Handling User Input and Displaying Chat History:**

* The handle\_userinput function is responsible for processing user input and displaying the chat history in the user interface.
* User input is passed to the conversation chain using st.session\_state.conversation, and the response is stored in the chat\_history.
* The conversation history is reversed and iterated over to display user and bot messages using HTML templates (bot\_avatar and user\_avatar) that include avatars and message content.

**OpenAI API Key:**

* The OpenAI API key is stored securely in a .env file, which is accessed using the load\_dotenv function from the dotenv library.
* The key is retrieved from the .env file and used to authenticate API calls to the OpenAI API.

**Main Function:**

* The main function serves as the entry point of the chatbot system, responsible for coordinating the setup and execution of the chatbot's functionalities.
* It starts by loading the environment variables securely from the .env file using the load\_dotenv function from the dotenv library. This includes accessing the OpenAI API key required for authentication during API calls.
* Next, the main function initializes the user interface using the Streamlit framework. It configures the page title and icon to enhance the visual appeal of the chatbot interface.
* The conversation and chat history variables are initialized in the Streamlit session state to maintain the conversation context between the user and the chatbot.
* User input is obtained through the st.text\_input function, allowing users to ask questions or enter queries about the college.
* The main function calls all the necessary functions to process the user input and generate appropriate responses. This includes calling the handle\_userinput function, which interacts with the conversation chain, passing the user's question, and retrieving the chatbot's response. The chat history is updated with the response obtained.
* Additionally, the main function calls the get\_pdf\_text function to extract text data from the PDF documents, the get\_text\_chunks function to split the extracted text into smaller chunks, and the get\_vectorstore function to create the vector store for efficient information retrieval.
* Furthermore, the main function sets up the conversation chain by calling the get\_conversation\_chain function, integrating the language model, the vector store converted into a retriever, and a conversation buffer memory.
* The reversed chat history is then iterated over to display the user and bot messages in the user interface. The bot\_avatar and user\_avatar HTML templates are used to render avatars and message content, providing a visually appealing chat-like interface.
* By calling the required functions within the main function, it ensures the seamless flow of conversation, handling user input, obtaining responses from the chatbot, and displaying the chat history in an interactive manner.

**4.3 ALGORITHM:**

The algorithm presented below provides an overview of the logic and steps involved in the functioning of our intelligent college assistant chatbot system. It outlines the sequence of actions and functions that are executed to process user input, retrieve information, and generate responses. The following is the pseudocode of the system's algorithmic flow:

Start Application

Load Libraries

Set Page Config

Define Session State

Display User Interface

Loop:

Handle User Input and Processing:

UserQuestion = GetUserInput()

ProcessUserQuestion(UserQuestion)

Extract Text from PDF Documents:

PDFDocs = GetPDFDocuments()

RawText = GetPDFText(PDFDocs)

Split Text into Chunks:

TextChunks = SplitText(RawText)

Generate Embeddings:

Embeddings = GenerateEmbeddings(TextChunks)

Create Vector Store:

VectorStore = CreateVectorStore(Embeddings)

Set up Conversation Chain:

ConversationChain = SetUpConversationChain(VectorStore)

Process User Input and Generate Responses:

Response = ProcessUserInput(ConversationChain)

Display Chat History and User Interface:

DisplayChatHistory(Response)

Continue Loop until Exit

End Application

**CHAPTER 5**

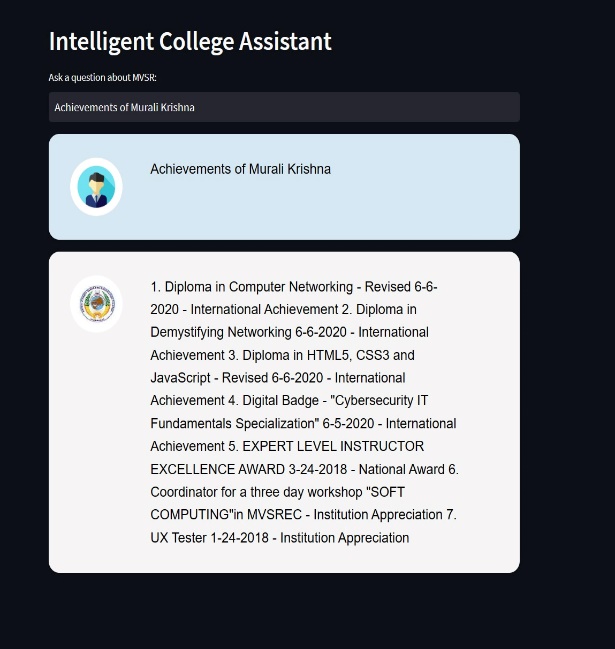
**TESTING AND RESULTS**

**5.1 TEST CASES:**

**TEST CASE 1: Question and answer flow**

1. **TEST CASE FOR ACHIVEMENTS QUERY**

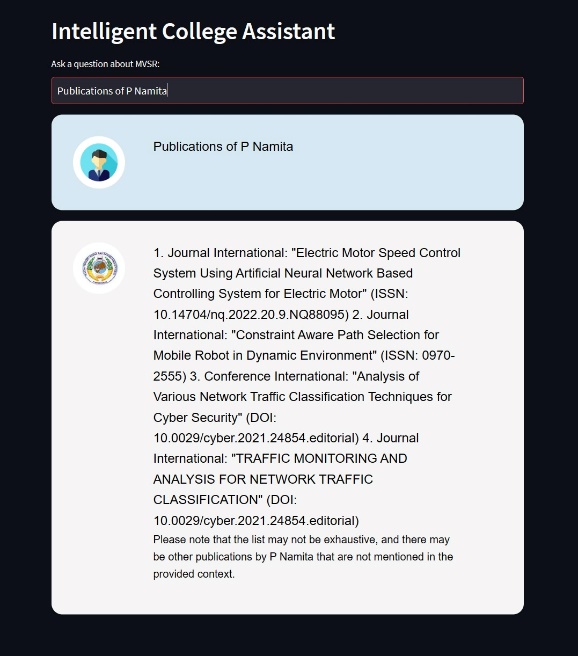
* **Test Query**: "Achievements of Murali Krishna sir"
* **Expected Outcome**: The chatbot should detect that the question is within its knowledge base and respond with a message indicating that the information requested is available. The chatbot should provide all the list of achievements of the specified faculty.
* **Actual Outcome:** Figure 5.1 shows the output when asked about the achievements of Murali Krishna sir.

****

*Figure 5.1 Achievements Query Result*

1. **TEST CASE FOR PUBLICATIONS QUERY**

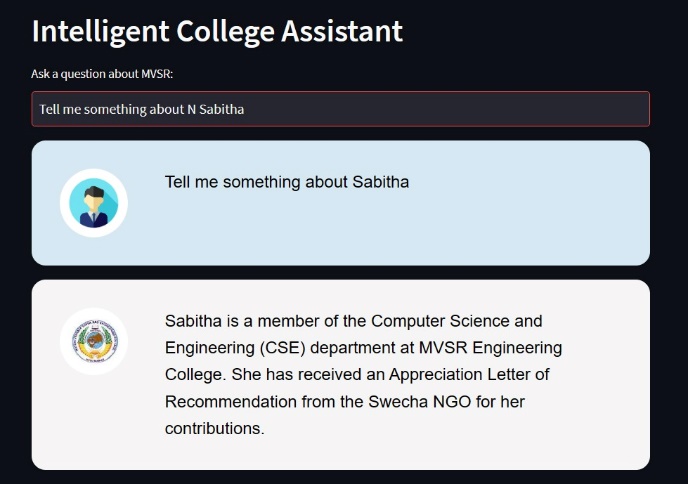
* **Test Query**: "Publications of P Namita mam"
* **Expected Outcome**: The chatbot should detect that the question is within its knowledge base and respond with a message indicating that the information requested is available. The chatbot should provide all the list of publications of the specified faculty from the knowledge base.
* **Actual Outcome:** Figure 5.2 shows the output when asked about the publications of P Namita mam.

****

*Figure 5.2 Publications Query Result*

1. **TEST CASE FOR ABOUT QUERY**

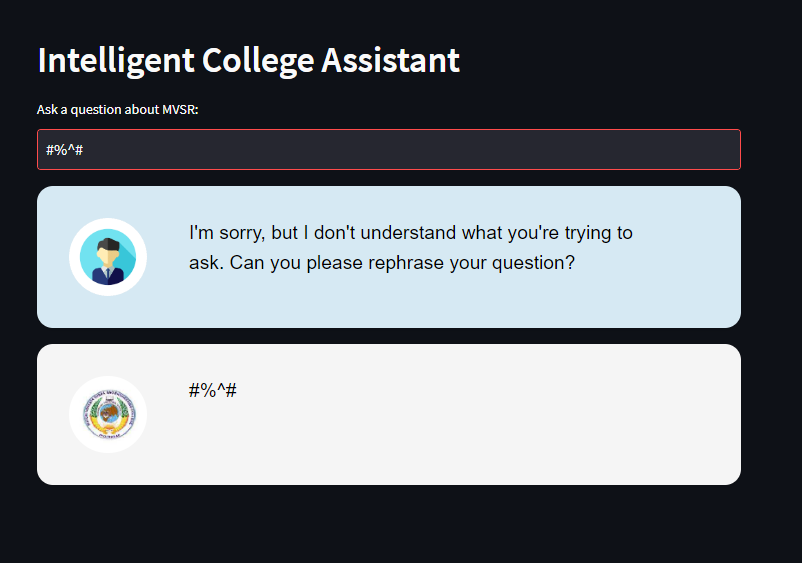
* **Test Query**: "Tell me something about N Sabitha mam"
* **Expected Outcome**: The chatbot should detect that the question is within its knowledge base and respond with a message indicating that the information requested is available. The chatbot should provide the information about the specific faculty which is present in the knowledge base.
* **Actual Outcome:** Figure 5.2 shows the output when asked about N Sabitha mam.

****

*Figure 5.3 About Query Result*

**TEST CASE 2: Input Validation**

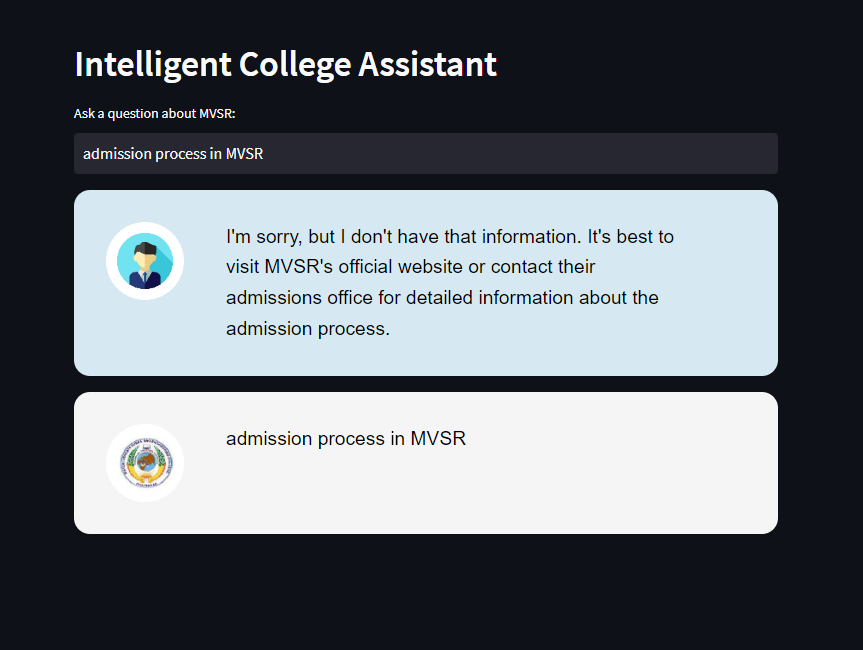
* **Test Query**: #%^#
* **Expected Outcome**: The chatbot should detect the invalid input and respond with a message prompting the user to enter a valid query or ask a proper question.
* **Actual Outcome**: As shown in figure 5.4 the chatbot correctly detects the invalid input and responds with a message requesting the user to enter a valid query or ask a proper question. In this case, when the user entered the input "#%^#", the chatbot displayed the following message: "I’m sorry, but I don’t understand what you’re trying to ask. Can you please rephrase your question"

****

*Figure 5.4 Chatbot response to invalid input*

**TEST CASE 3: Out-of-scope question**

* **Test Query**: "admission process in MVSR"
* **Expected Outcome**: The chatbot should detect that the question is not within its knowledge base and respond with a message indicating that the information requested is not available. The chatbot should also provide guidance on where the user can find accurate information about the admission process, such as directing them to the college website or relevant admission resources.
* **Actual Outcome:** Figure 5.5 shows the output when asked about the admission process in MVSR.



*Figure 5.5 Chatbot response to out-of-scope admission process question*

**CHAPTER 6**

**CONCLUSION & FUTURE ENHANCEMENTS**

**CONCLUSION**

In conclusion, the development and implementation of our college assistant chatbot have been a resounding success. Through the utilization of advanced AI technologies, we have created a powerful tool that greatly enhances the user experience and facilitates efficient access to faculty information at MVSR.

Our chatbot has revolutionized the way users interact with the college's resources by providing instant and accurate responses to their queries. With just a few clicks or by asking a question in natural language, users can easily retrieve faculty achievements, publications, and memberships, eliminating the need for manual search and saving valuable time and effort.

One of the key achievements of our project is the improvement in accessibility. The chatbot's conversational interface allows users to interact with it in a user-friendly and intuitive manner, regardless of their technical expertise. It has made information about MVSR readily available 24/7, ensuring that students, faculty, and other stakeholders can obtain the required information at their convenience.

Looking ahead, we envision exciting future possibilities for our chatbot. With continued development, we plan to incorporate advanced natural language processing techniques, personalize user experiences, and integrate with additional college systems and data sources. These enhancements will further solidify the chatbot's position as an indispensable tool for the college community.

**FUTURE ENHANCEMENTS**

**Integration with External Systems**: Integrate the chatbot with other college systems or databases to provide seamless access to additional information, such as course schedules, event calendars, or admission details. This can enhance the chatbot's usefulness as a comprehensive college assistant.

**Rich Media and Visual Responses:** Enhance the chatbot's responses by incorporating rich media elements such as images, videos, and interactive visualizations. This can make the interaction more engaging and facilitate better communication of information.

**Natural Language Processing (NLP) Enhancements:** Explore advanced NLP techniques to improve the chat bot's understanding and response generation capabilities. This could include sentiment analysis, entity recognition, and intent classification to provide more accurate and contextual responses.

**REFERENCES**

**[1]**PyPDF2libraryReference:"PyPDF2-PythonPDFLibrary." https://github.com/mstamy2/PyPDF2

**[2]**LangChainlibrary:Reference:"LangChain-ConversationalAILibrary." https://python.langchain.com/

**[3]** OpenAI GPT and OpenAI Embeddings Citations: "Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners.OpenAIBlog.","OpenAIEmbeddings-OpenAI." https://github.com/openai/embedding-models

**[4]** FAISS library Citation: "Johnson, J., Douze, M., & Jégou, H. (2017). Billion-scale similarity search with GPUs. arXiv preprint arXiv:1702.08734."

**[5]** Streamlit library Reference: "Streamlit - The fastest way to build custom ML tools." https://www.streamlit.io/

**APPENDIX**

**Source code:**

**.env:**

OPENAI\_API\_KEY="api key here**”**

**Template.py**

template = '''

<style>

.chat-message {

padding: 2rem;

border-radius: 1rem;

margin-bottom: 1rem;

display: flex;

box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

}

.chat-message.user {

background-color: #d6e9f3;

}

.chat-message.bot {

background-color: #f5f5f5;

}

.chat-message .avatar {

width: 15%;

}

.chat-message .avatar img {

max-width: 78px;

max-height: 78px;

border-radius: 50%;

object-fit: cover;

}

.chat-message .message {

width: 80%;

padding: 0 1.5rem;

color: #000;

font-family: Arial, sans-serif;

font-size: 1.2rem;

}

</style>

'''

bot\_avatar = '''

<div class="chat-message bot">

<div class="avatar">

<img src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTzGQJP3QhW4fwxM\_AHrWtdkXYDCnUFkz57IzGnuC1ftk8FKg1g7J9tpUOhhHoI9AQZujo&usqp=CAU" style="max-height: 78px; max-width: 78px; border-radius: 50%; object-fit: cover;">

</div>

<div class="message">{{MSG}}</div>

</div>

'''

user\_avatar = '''

<div class="chat-message user">

<div class="avatar">

<img src="https://e7.pngegg.com/pngimages/114/356/png-clipart-time-student-recruitment-learning-professional-others-service-vector-icons.png" style="max-height: 78px; max-width: 78px; border-radius: 50%; object-fit: cover;">

</div>

<div class="message">{{MSG}}</div>

</div>

**app.py:**

import streamlit as st

from dotenv import load\_dotenv

from PyPDF2 import PdfReader

from langchain.text\_splitter import CharacterTextSplitter

from langchain.embeddings import OpenAIEmbeddings

from langchain.vectorstores import FAISS

from langchain.chat\_models import ChatOpenAI

from langchain.memory import ConversationBufferMemory

from langchain.chains import ConversationalRetrievalChain

from Templates import template, bot\_avatar, user\_avatar

def get\_pdf\_text(pdf\_docs):

text = ""

for pdf in pdf\_docs:

pdf\_reader = PdfReader(pdf)

for page in pdf\_reader.pages:

text += page.extract\_text()

return text

def get\_text\_chunks(text):

text\_splitter = CharacterTextSplitter(

separator="\n",

chunk\_size=1000,

chunk\_overlap=200,

length\_function=len

)

chunks = text\_splitter.split\_text(text)

return chunks

def get\_vectorstore(text\_chunks):

embeddings = OpenAIEmbeddings()

vectorstore = FAISS.from\_texts(texts=text\_chunks, embedding=embeddings)

return vectorstore

def get\_conversation\_chain(vectorstore):

llm = ChatOpenAI()

memory= ConversationBufferMemory(memory\_key='chat\_history', return\_messages=True)

conversation\_chain= ConversationalRetrievalChain.from\_llm(

llm=llm,

retriever=vectorstore.as\_retriever(),

memory=memory

)

return conversation\_chain

def handle\_userinput(user\_question):

response = st.session\_state.conversation({'question': user\_question})

st.session\_state.chat\_history = response['chat\_history']

reversed\_chat\_history= reversed(st.session\_state.chat\_history) # Reverse the chat history

for i, message in enumerate(reversed\_chat\_history):

if i % 2 == 0:

st.write(user\_avatar.replace("{{MSG}}", message.content), unsafe\_allow\_html=True)

else:

st.write(bot\_avatar.replace("{{MSG}}", message.content), unsafe\_allow\_html=True)

def main():

load\_dotenv()

st.set\_page\_config(page\_title="Intelligent College Assistant", page\_icon="🤖")

st.write(template, unsafe\_allow\_html=True)

if "conversation" not in st.session\_state:

st.session\_state.conversation = None

if "chat\_history" not in st.session\_state:

st.session\_state.chat\_history = None

st.header("Intelligent College Assistant ")

user\_question = st.text\_input("Ask a question about MVSR:")

if user\_question:

handle\_userinput(user\_question)

# Specify the path or file name of the PDF file

pdf\_file\_path = "data/finaldata.pdf"

pdf\_docs = [pdf\_file\_path]

# get pdf text

raw\_text = get\_pdf\_text(pdf\_docs)

# get the text chunks

text\_chunks = get\_text\_chunks(raw\_text)

# create vector store

vectorstore = get\_vectorstore(text\_chunks)

# create conversation chain

st.session\_state.conversation= get\_conversation\_chain(vectorstore)

if \_\_name\_\_ == '\_\_main\_\_':

main()