

# **College Enquiry Chatbot**

## **A PROJECT REPORT**

**Submitted By**

**Brahm Dev Pandey - University Roll No: 2100290140052**

**Anuj - University Roll No: 2100290140031**

**Atif Ali - University Roll No: 2100290140040**

**Submitted in partial fulfilment of the  
Requirements for the Degree of**

## **MASTER OF COMPUTER APPLICATION**

**Under the supervision of**

**Ms. Neelam Rawat & Mr. Prashant Agrawal**

**Associate Professor**



**DEPARTMENT OF COMPUTER APPLICATIONS**

**KIET GROUP OF INSTITUTIONS, Ghaziabad  
Uttar Pradesh-201206**

**(June 2023)**

## DECLARATION

I hereby declare that the work presented in report entitled “College Enquiry Chatbot” was carried out by me. I have not submitted the matter embodied in this report for the award of any other degree or diploma of any other University or Institute. I have given due credit to the original authors/sources for all the words, ideas, diagrams, graphics, computer programs, that are not my original contribution. I have used quotation marks to identify verbatim sentences and give credit to the original authors/sources. I affirm that no portion of my work is plagiarized, and the experiments and results reported in the report are not manipulated. In the event of a complaint of plagiarism and the manipulation of the experiments and results, I shall be fully responsible and answerable.

**Name:** Brahm Dev Pandey  
**Roll No.:** 2100290140052

**(Candidate Signature)**

**Name:** Atif Ali  
**Roll No.:** 2100290140040

**(Candidate Signature)**

**Name:** Anuj  
**Roll No.:** 2100290140031

**(Candidate Signature)**

## **CERTIFICATE**

Certified that **Brahm Dev Pandey (210290140052)**, **Atif Ali (2100290140040)**, **Anuj (2100290140031)** have carried out the project work “**College Enquiry Chatbot**” for Master of Computer Applications from Dr. A.P.J. Abdul Kalam Technical University Technical University (AKTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the students themselves and the contents of the project report do not form the basis for the award of any other degree to the candidates or to anybody else from this or any other University/Institution.

**Brahm Dev Pandey (2100290140052)**

**Atif Ali (2100290140040)**

**Anuj (2100290140031)**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:

**Ms. Neelam Rawat**

**(Associate Professor)**

**Mr. Prashant Agrawal**

**(Associate Professors)**

**Department of Computer Applications  
KIET Group of Institutions, Ghaziabad**

**Signature of Internal Examiner**

**Signature of External Examiner**

**Dr. Arun Kumar Tripathi  
Head, Department of Computer Applications  
KIET Group of Institutions, Ghaziabad**

## ABSTRACT

Potential students' correspondence with a college department is handled manually, which takes a lot of time. It is highly important to have one-on-one interactions with people. Unfortunately, due to the large number of applications received each year, one-on-one interactions are rarely possible. A member of the academic staff will need to dedicate many hours to the communication in order to contact each student and discover appropriate answers. His costs and time might be cut, which would be beneficial.

By creating a persuasive chatbot, the project hopes to lighten the load on the head of admissions and perhaps other users. In order to search through the collection of data and maybe locate an answer, an appropriate algorithm must be created. If the user is not pleased with the response, the computer responds and offers a pertinent web link. Users also have access to a web interface.

The project's accomplishments can be summed up as follows. A literature review was conducted to prepare the project's backdrop. The system's criteria were created, and a variety of methods and tools, including keyword and template matching, were looked into. The technique employed combines string similarity with keyword matching. The suggested algorithm has been implemented in a workable system. The system was assessed using input from prospective students who utilised it as well as question and answer records.

## ACKNOWLEDGEMENTS

Success in life is never attained single handedly. My deepest gratitude goes to my thesis supervisor, Ms. Neelam Rawat (Associate Professor) and Mr. Prashant Agrawal (Associate Professor) for their guidance, help and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions have guided me a lot in completing this project successfully.

Words are not enough to express my gratitude to Dr. Arun Kumar Tripathi, Professor and Head, Department of Computer Applications, for his insightful comments and administrative help at various occasions. Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

**Brahm Dev Pandey (2100290140052)**

**Atif Ali (2100290140040)**

**Anuj (2100290140031)**

## TABLE OF CONTENTS

Declaration	ii
Certificate	iii
Abstract	iv
Acknowledgements	v
Table of contents	vi
List of Figures	viii
Chapter 1: Introduction	1
1.1 Overview	1
1.2 Problem Statement	2
1.3 Objectives	2
1.4 Scope	3
1.5 Hardware / Software used in project	5
Chapter 2: Literature Review	6
Chapter 3: Software development life cycle model	11
Chapter 4: Feasibility Study	13
4.1 Technical feasibility	13
4.2 Operational feasibility	14
4.3 Economic feasibility	14
4.4 Behavioural feasibility	15
Chapter 5: System requirements	16
5.1 Functional requirements	16
5.2 Non-functional requirements	16
Chapter 6: Background and alternatives	18
6.1 Chatbot	18
6.2 Types of chatbot	18
6.3 How do chatbot works?	19
6.4 Options to build a chatbot	21
6.5 Chatbot platform alternatives	22
6.6 Selected platforms	23
6.7 How to setup RASA and start a chatbot	24
Chapter 7: Models and Interfaces	26
7.1 Basics of Rasa Opensource Conversational AI	26
7.2 Architectural Model	27

7.3 Rasa framework files	29
7.4 Flowchart and Sequence diagrams	31
Chapter 8: System demonstration	34
8.1 Implementation details	34
8.2 Implementation issues	34
8.3 Important information	35
8.4 Snapshots	36
8.5 Code	44
Chapter 9: Modules Description	66
Chapter 10: Testing	67
Chapter 11: Scope for further improvements	71
References	73

## LIST OF FIGURES

Figure No.	Name of Figure	Page No.
1.1	Schematic diagram of chatbot	3
3.1	Waterfall model	11
6.1	Python download screen	24
6.2	Testing python version	24
6.3	Rasa bot initial chatting	25
7.1	Basics of Rasa Opensource Conversational AI	27
7.2	Architectural Model	28
7.3	nul.yml file content	29
7.4	domain.yml and nlu.yml file content	30
7.5	domain.yml and stories.yml file content	30
7.6	Flowchart of chatbot	32
7.7	Sequence diagram of user	33
7.8	Sequence diagram of developer	33
8.1	Rasa training log screen	37
8.2	Rasa action server terminal screen	38
8.3	Rasa server screen	40
8.4	Starting the frontend	40
8.5	Chatbot interaction screen 1	41
8.6	Chatbot interaction screen 2	42
8.7	Chatbot interaction screen 3	43



# CHAPTER 1: INTRODUCTION

## 1.1 Overview

Although the admissions procedure currently in place is effective, getting in touch with a college staff person is exceedingly time-consuming and challenging. Yet, if the applicant could speak to a convincing chatbot, able to address their concerns with information on admissions, hostel amenities, paying fees in instalments, and what courses are offered, the problem might be partially resolved.

The chatbot should be able to communicate with a user in a way similar to the following:

**User:** Hello!

**ChatBot:** Hello, how can I help you?

**User:** What are the courses offered by your college?

**ChatBot:** KIET provides B.Tech, B.Pharma, MCA, MBA, M.Pharma

**User:** What is the percentage criteria to take admission in MCA?

**ChatBot:** Student must have Bachelor's degree with minimum 60 percentage and with no backlog.

**User:** What are the fees for MCA?

**ChatBot:** Total yearly fees of MCA are Rs. 1,38,500 /-

**User:** Is there any concession for economically weaker students?

**ChatBot:** Meritorious students can get fee waiver seats and also students can fill UP Scholarship form.

**User:** Thank you!

**ChatBot:** Thanks for visiting our website. Have a nice day!

The goal of this project is to build a chatbot that students may utilise to quickly get answers to their questions from the college website. A chatbot is a computer programme that can conduct real conversations via text and/or audio. Chatbots can mimic human talks using artificial intelligence (AI). The two types of chatbots are as follows. Chatbots that operate under commands and use a database of possible responses and heuristics fall under one type. In order for the bot to respond, the user's questions must be highly detailed. As a result, these bots can only respond to a specific set of queries and are hence limited in what they can do. The second group consists of chatbots powered by artificial intelligence (AI) or machine learning algorithms. These bots can respond to confusing inquiries, so the user does. Every time a user asks a question, the bot first analyses the question, then finds entities and intents, creates a response, and sends it back to the user.

The necessity for traditional websites to include a chat feature, where a bot is needed to be able to talk with users and address their questions, is what drives the development of AI-powered chatbots. Chatbots can work without an upper limit, which truly scales up the operations while a live person can only manage two to three activities at once. Additionally, if a company or institution receives a lot of inquiries, the stress on the customer service team is reduced by having a chatbot on the website. Comparing a chatbot to a human support team, it is obvious that the response rate is improved. Additionally, since millennials prefer live chats to phone calls, they are drawn to chatbots since they offer a highly participatory marketing platform. A chatbot can also automate the routine tasks.

## **1.2 Problem Statement**

Registration for new students in the college numerous fields occurs at the beginning of each semester, and phone calls for admission and registration are frequent. As a result, the workload and workload for the staff of the Deanship of admission and registration rise. As a result of the families and individuals who seek to register pouring into the Deanship prevents the staff from responding to social media and phone calls.

## **1.3 Objectives**

- Enhance the user experience: A college admission chatbot can provide a more personalized and interactive experience for prospective students and their families, answering their questions and guiding them through the admission process.
- Increase efficiency: By automating certain tasks, such as answering frequently asked questions and providing information about the college's programs and policies, the chatbot can free up admissions staff to focus on more complex or strategic tasks.
- Improve accessibility: A chatbot can be available 24/7 and accessible from any device with an internet connection, making it easier for prospective students to get the information they need, regardless of their location or time zone.
- Boost engagement: Chatbots can engage with prospective students in a way that feels more natural and conversational than static web pages or brochures, potentially increasing their interest in the college and encouraging them to apply.
- Generate leads: By capturing contact information and preferences from users, the chatbot can help the college build a pipeline of prospective students and target its recruitment efforts more effectively.
- Provide data insights: By tracking user interactions with the chatbot, the college can gain insights into the questions and concerns of prospective students and use that information to improve its recruitment and admissions strategies.

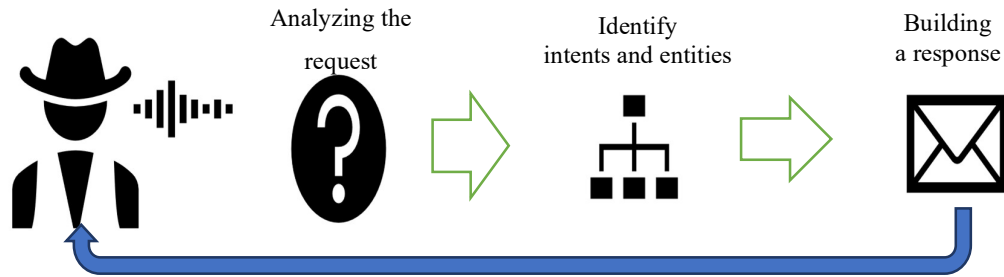


Fig. 1.1: Schematic diagram of chatbot

- **Analysing the user request:**

This initial step involves receiving and understanding the user's input or request. The chatbot system processes the user's message to determine its intent, which represents the goal or purpose behind the user's query. The analysis may involve techniques like natural language understanding (NLU) to comprehend the user's message accurately.

- **Identifying intents and entities:**

Once the user's request is analysed, the chatbot system identifies the intent and extracts any relevant entities from the user's input. Intents represent the high-level goal or action the user wants to perform, while entities are specific pieces of information within the user's message that are required to fulfil the user's intent. For example, if the user asks, "Who is the HOD of MCA department?", the intent could be "hod\_enquiry" with the entity "MCA" representing the department.

- **Building a response:**

With the user's intent and entities identified, the chatbot system generates an appropriate response. This step involves formulating a reply or providing relevant information to answer the user's query. The response can be generated using natural language generation (NLG) techniques to convert the system's output into human-readable text that the user can understand.

## 1.4 Scope

The scope of this project encompasses the development, implementation, and initial deployment of a college enquiry chatbot to streamline the admission process and provide timely and accurate information to prospective students. The following aspects are within the scope of this project:

- **Enquiry Handling:** This chatbot is designed to handle admission-related enquiries, including queries about admission requirements, application procedures, course details, deadlines, and general information about the college.

- **User Interface and Experience:** This chatbot have a user-friendly interface allowing prospective students to interact with it easily. This chatbot interface is accessible through a web-based platform, ensuring compatibility with popular web browsers and devices.
- **Providing accurate and timely information:** The chatbot will serve as a reliable source of information, answering frequently asked questions about the college, academic programs, admission requirements, campus facilities, and more.
- **Assisting with admissions:** The chatbot will guide prospective students through the admission process, helping them with application forms, deadlines, required documents, and providing updates on the status of their applications.
- **Target Audience or Users:** The chatbot will cater to various stakeholders within the college community, including:
  - **Prospective students:** Individuals who are interested in applying to the college and require information about academic programs, admission criteria, scholarships, campus life, and more.
  - **Current students:** Enrolled students seeking information on course registration, academic calendars, campus events, student services, and other campus-related inquiries.
  - **Parents and guardians:** They may have questions about tuition fees, financial aid, campus safety, accommodation options, and the overall college experience for their children.
  - **Faculty members:** They may require information about administrative processes, course schedules, academic resources, or support services available to students.

## 1.5 Hardware / Software Requirements for development

### Hardware Requirements

S. N.	Description
1	PC with 10 GB or more Hard disk.
2	PC with 2 GB RAM.
3	PC with core i3 or above processor.

### Software Requirements

S. N.	Description	Type
1	Operating System	Windows 10 or 11 or Ubuntu 18.04 or above
2	Language	Python 3
3	Front End	React 17
4	IDE	Google Colab, VS Code
5	Browser	Chrome, Firefox, Edge

## CHAPTER 2: LITERATURE REVIEW

In recent years, chatbots have gained significant popularity in the education sector. College admission chatbots are now being implemented by many educational institutions as a means of engaging with prospective students, providing them with relevant information, and guiding them through the admissions process. This literature review contains the various studies that have been conducted on the use of college admission enquiry chatbots.

**1. “Natural Language Processing in Artificial Intelligence” by Brojo Kishore**

The aim of this edited book is to foster interactions among researchers and practitioners in NLP, AI, and allied areas. The edited book covers theoretical work, advanced applications, approaches, and techniques for computational models of information and its presentation by language (artificial, human, or natural in other ways). The goal is to promote intelligent natural language processing (NLP) and related models of thought, mental states, reasoning, and other cognitive processes.

**2. “Graph-based Natural Language Processing and Information Retrieval”**

This book is a comprehensive description of the use of graph-based algorithms for natural language processing and information retrieval. It brings together areas as diverse as lexical semantics, text summarization, text mining, ontology construction, text classification, and information retrieval, which are connected by the common underlying theme of using graph-theoretical methods for text-and information-processing tasks.

**3. “Automation, Innovation and Work”**

This book explores the impact that these intelligent robots and intelligent in formats will have on social and societal development. The author tackles the question of singularity from three distinct standpoints: technological singularity – the intelligence of machines compared to that of humans – which he argues will bring about a qualitatively new labour market; economic singularity – the consequences for work relationships, value creation and employment – which he asserts will promote full automation, result in precarious contracts with low salaries, and, in some countries, possibly lead to the introduction of a universal basic income; and social singularity – the consequences of technological and economic singularity for democratic processes, bureaucratic procedures for exercising authority and control, and the direction in which society will develop.

**4. “Applied Machine Learning for Smart Data Analysis” by N Dey**

Applied machine learning for smart data analysis is a practical guide for anyone interested in using machine learning techniques to analyse and understand data. The book covers the basics of machine learning, including data preparation,

feature selection, and model evaluation. It also includes in-depth discussions of popular machine learning algorithms such as decision trees, logistic regression, and support vector machines. The book is designed to be accessible to readers with a basic understanding of statistics and programming, and includes examples and case studies from a variety of fields, including finance, healthcare, and social media. Overall, "Applied Machine Learning for Smart Data Analysis" is a useful resource for anyone looking to use machine learning to gain insights from data.

**5. “Adversarial Machine Learning” by D. Joseph**

Adversarial Machine Learning is a comprehensive guide that explores the field of adversarial machine learning, which is concerned with the vulnerability of machine learning models to malicious attacks. The book covers a wide range of topics, including different types of attacks that can be launched against machine learning models, such as evasion and poisoning attacks. It also discusses the defence mechanisms that can be employed to mitigate these attacks, such as adversarial training, data sanitization, and model verification. The book is intended for readers with a strong background in machine learning and is well-suited for researchers, academics, and practitioners interested in the security and robustness of machine learning models. Overall, "Adversarial Machine Learning" provides a comprehensive overview of this emerging field and is a valuable resource for those looking to gain a deeper understanding of the potential vulnerabilities and defences in machine learning systems.

**6. “Design of Intelligent Applications using Machine Learning and Deep Learning Techniques” edited by Ramchandra Sharad Mangrulkar**

This book is a comprehensive guide that provides insights into the design and development of intelligent applications using machine learning and deep learning techniques. The book covers a broad range of topics, including data preparation, feature selection, model selection, and evaluation, as well as deep learning techniques such as convolutional neural networks and recurrent neural networks.

The book includes contributions from a variety of experts in the field and covers applications in diverse fields such as healthcare, finance, and marketing. The authors provide practical examples and case studies to illustrate how machine learning and deep learning can be used to build intelligent applications that can make predictions, classify data, and automate tasks.

**7. “Supervised Machine Learning: Optimization Framework and Applications with SAS and R” by Tanya Kolosova and Samuel Berestizhevsky**

This book is a practical guide that provides a comprehensive overview of supervised machine learning and its applications using the SAS and R programming languages. The book covers the foundational concepts of supervised machine learning, including regression, classification, and decision trees.

The authors provide detailed explanations of optimization frameworks and how they can be applied to various supervised machine learning algorithms, such as linear regression, logistic regression, and support vector machines. The book also covers practical applications of supervised machine learning in various fields, such as finance, healthcare, and marketing.

**8. "Practical Guide to Logistic Regression" by Joseph M. Hilbe**

This book is a comprehensive guide that provides practical insights into logistic regression, a popular statistical method used for analysing binary and categorical data. The book covers the foundational concepts of logistic regression, including model formulation, estimation, and model evaluation.

The book is designed for readers with a basic understanding of statistics and regression analysis and provides a useful resource for researchers, students, and practitioners interested in using logistic regression for data analysis.

**9. "Bayesian Modeling and Computation in Python" by Osvaldo A. Martin, Ravin Kumar, Junpeng Lao**

The book is designed for readers with a basic understanding of statistics and programming and provides a useful resource for researchers, students, and practitioners interested in using Bayesian modelling and computation for data analysis. Overall, "Bayesian Modelling and Computation in Python" is a valuable resource for those looking to gain a deeper understanding of Bayesian modelling and its practical applications using the Python programming language.

**10. "Data Science and Analytics with Python" by Jesus Rogel-Salazar**

is a comprehensive guide that provides practical insights into data science and analytics using the Python programming language. The book covers the foundational concepts of data science, including data cleaning, data visualization, and exploratory data analysis.

The author provides a step-by-step approach to building and interpreting machine learning models using various Python libraries, including NumPy, Pandas, Scikit-Learn, and Matplotlib. The book also covers practical applications of data science and analytics in various fields, such as finance, healthcare, and social sciences.

**11. "Chatbot for college website" by Kumar Shivam, Khan Saud, Manav Sharma, Saurav Vashishth, Sheetal Patil**

The chatbot created by Kujar Shivam, Khan Saud, Manav Sharma, and Saurabh Vashishth for the college website is an AI-powered tool that offers an interactive and efficient way to access information and assistance. Users can easily navigate different sections such as admissions, academics, faculty, facilities, and events. The chatbot utilizes natural language processing to understand user queries and provide quick and accurate responses. It assists with tasks like guiding prospective students through the application process and helping current students access schedules and grades. The user-friendly interface features a chat window where



users can type queries and receive instant responses, along with suggestions for easier navigation. Feedback is encouraged to improve the chatbot's performance, allowing the developers to enhance its functionality. Overall, the chatbot improves the user experience by offering personalized and convenient access to information, making it a valuable tool for the college community.

**12. Software Testing and Continuous Quality Improvement By William E. Lewis**

The book delves into various testing methodologies, techniques, and tools that can be employed to achieve comprehensive and effective software testing. It covers both manual and automated testing approaches, discussing their strengths, limitations, and appropriate usage scenarios. Lewis also emphasizes the significance of test case design, test coverage, and test management in achieving high-quality software products.

Furthermore, the book emphasizes the concept of continuous quality improvement, which involves an ongoing effort to enhance software development practices and deliver superior products. Lewis explains how to establish quality metrics, perform root cause analysis, and implement corrective actions to continuously improve the quality of software products and processes.

Throughout the book, Lewis provides practical insights, examples, and case studies to illustrate the principles and best practices of software testing and continuous quality improvement. The aim is to equip software professionals, testers, and quality assurance teams with the necessary knowledge and techniques to effectively ensure the quality and reliability of software systems.

**13. Knowledge Driven Development Bridging Waterfall and Agile Methodologies by Manoj Kumar Lal**

The author provides practical insights and techniques for incorporating knowledge management into the software development lifecycle. He explains how to capture, share, and utilize knowledge effectively, enabling teams to make informed decisions, reduce rework, and improve overall project outcomes. Lal also discusses the importance of fostering a culture of continuous learning and knowledge sharing within development teams.

Through real-world examples, case studies, and practical advice, Lal demonstrates how the knowledge-driven approach can enhance collaboration, enable more effective communication, and facilitate smoother transitions between project phases. The book serves as a guide for software professionals and project managers seeking to optimize their development processes by leveraging knowledge as a strategic asset.

**14. “An Analytical Study and Review of open Source Chatbot framework, RASA” by Rakesh Kumar Sharma & Manoj Joshi**

The paper titled "An Analytical Study and Review of open Source Chatbot framework, RASA" by Rakesh Kumar Sharma and Manoj Joshi provides a comprehensive analysis and evaluation of the RASA chatbot framework, which is

an open-source platform for building conversational AI applications. The authors aim to explore the features, capabilities, and overall performance of RASA and provide insights into its strengths and limitations.

The paper also evaluates RASA's performance based on various criteria such as accuracy, scalability, and ease of deployment. The authors discuss the availability of pre-trained models, the community support, and the documentation provided by RASA, which contribute to its usability and accessibility.

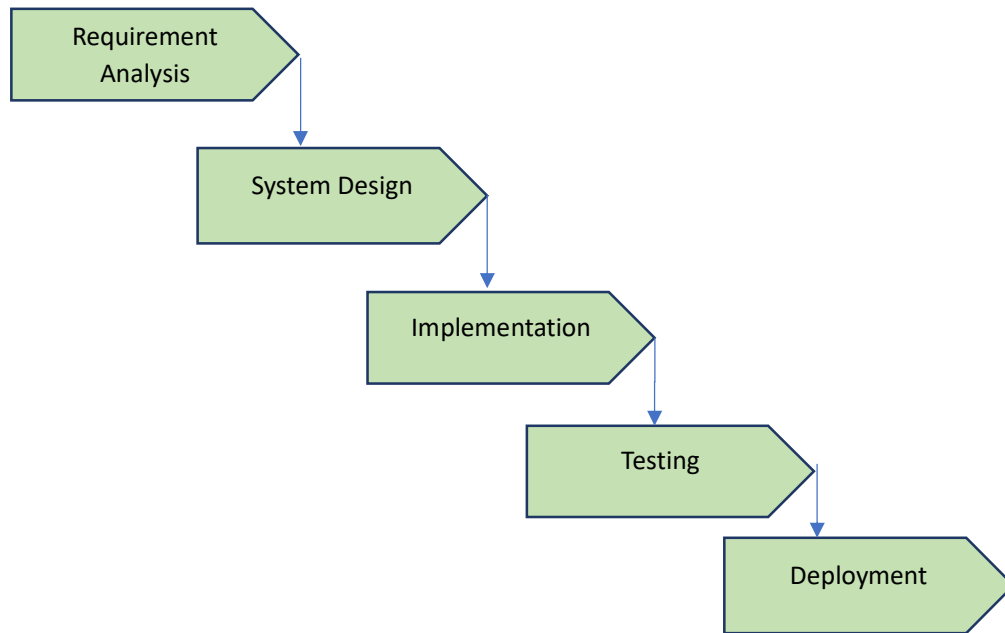
**15. “College Enquiry Chatbot using Rasa Framework” by Siddhant Meshram, Namit Naik, Megha VR, Tanmay More & Shubhangi Kharche**

The paper presents a practical implementation of a college enquiry chatbot using the Rasa framework. It showcases the capabilities of Rasa in building conversational agents and demonstrates its potential in automating college-related enquiries. The study serves as a valuable resource for researchers and developers interested in utilizing Rasa for developing similar chatbot applications.

It also presents a study on the development of a college enquiry chatbot using the Rasa framework. The authors aim to create an intelligent conversational agent that can assist students with their queries related to college admissions, courses, and other relevant information.

## CHAPTER 3: SDLC MODEL

Chatbots have emerged as a valuable tool across various industries, including education, providing efficient and personalized assistance to users. Recently, I had the opportunity to develop a college enquiry chatbot using the waterfall model, integrating Rasa as the backend framework and ReactJS as the frontend framework.



**Fig. 3.1: Waterfall model**

### **Why use the Waterfall Model:**

The waterfall model is a sequential software development approach that follows a linear progression of phases, starting from requirements gathering and ending with deployment. This model was chosen for the development of the college enquiry chatbot due to its clear structure, defined stages, and suitability for projects with well-defined requirements. By following the waterfall model, I was able to ensure a systematic and organized development process.

### **Requirements Gathering:**

The first phase of the waterfall model involved gathering and analysing the requirements for the college enquiry chatbot. I collaborated with college administrators, faculty members, and students to identify common queries related to admissions, courses, deadlines, and campus facilities. This allowed me to gain a comprehensive understanding of the users' needs and expectations.

**System Design:**

Once the requirements were established, I proceeded to design the system architecture and the conversational flow of the chatbot. This phase involved defining the intents, entities, and actions in Rasa, as well as creating the UI components in ReactJS. The system design stage ensured that all the necessary components and functionalities were properly defined before proceeding to the next phase.

**Implementation:**

With the system design in place, I began the implementation phase. Using Rasa as the backend framework, I trained the language model with Rasa NLU, enabling it to understand user intents and extract relevant entities from queries. I implemented custom actions and integrated external APIs to fetch real-time data. In parallel, using ReactJS as the frontend framework, I built the user interface, including chat bubbles, input forms, and buttons. The implementation phase focused on developing the core functionality of the chatbot.

**Testing and Quality Assurance:**

Once the implementation was completed, I conducted comprehensive testing to ensure the chatbot's functionality, reliability, and accuracy. This involved unit testing individual components, integration testing to ensure proper communication between the frontend and backend, and user acceptance testing to evaluate the chatbot's performance in realistic scenarios. Feedback from users and stakeholders played a vital role in identifying and addressing any issues or improvements required.

**Deployment and Maintenance:**

After successfully testing the chatbot, it was ready for deployment. I deployed the chatbot to a cloud platform, making it accessible to users. Continuous monitoring and maintenance were implemented to ensure its smooth operation and address any potential issues that arose.

**Conclusion:**

By utilizing the waterfall model, I was able to develop a college enquiry chatbot that efficiently addressed common queries and enhanced the user experience. The sequential nature of the waterfall model allowed for a structured development process, ensuring that each phase was completed before moving on to the next. The integration of Rasa as the backend framework and ReactJS as the frontend framework provided the necessary tools for effective language understanding, dialogue management, and user interface development. The success of this project demonstrated the benefits of using the waterfall model in developing chatbot applications.

## CHAPTER 4: FEASIBILITY STUDY

To conduct the feasibility study for the college admission enquiry chatbot project, a systematic approach was followed to evaluate the technical, operational, and economical aspects. The study aimed to assess the viability of developing and implementing a chatbot system to handle inquiries related to college admissions.

### 4.1 Technical Feasibility

The technical feasibility analysis focused on evaluating the key components required for building an effective chatbot system. The technical feasibility of a college enquiry chatbot is robust, thanks to advancements in natural language processing (NLP), machine learning, and artificial intelligence (AI) technologies. NLP algorithms enable the chatbot to understand and interpret human language, allowing it to accurately comprehend and respond to user enquiries. This ensures effective communication between the chatbot and users, mimicking human-like interactions.

Machine learning techniques play a crucial role in the technical feasibility of a chatbot. Through training and continuous learning from user interactions, the chatbot can improve its responses over time. It can recognize patterns, understand user preferences, and provide increasingly accurate and personalized information. This adaptability enhances the chatbot's ability to cater to a wide range of user enquiries effectively.

Furthermore, the integration of a college enquiry chatbot with existing systems and databases is technically feasible. The chatbot can access and retrieve information from various sources, such as course catalogues, admission databases, and campus facilities databases. This integration ensures that the chatbot provides up-to-date and relevant information to users, enhancing the overall user experience.

Additionally, the deployment of a chatbot can be implemented across multiple platforms and channels, including websites, messaging apps, and social media platforms. This allows users to access the chatbot using their preferred communication channel, increasing convenience and accessibility. Technical feasibility also extends to the ability to handle a large volume of simultaneous enquiries, ensuring scalability and efficient processing of user requests.

Moreover, advancements in cloud computing technologies provide the necessary infrastructure to support the technical feasibility of a chatbot. Cloud-based hosting and computing resources enable the chatbot to handle high volumes of traffic and ensure reliable and seamless performance.

In conclusion, the technical feasibility of a college enquiry chatbot is well-established. Leveraging NLP, machine learning, AI, and cloud computing technologies, a chatbot can effectively understand user queries, learn from interactions, integrate with existing systems, and provide accurate and personalized responses. These technical capabilities make a college enquiry chatbot a practical and valuable tool for enhancing the enquiry process within a college or university setting.

## **4.2 Operational Feasibility**

The operational feasibility analysis focused on assessing user acceptance and the practicality of deploying a college admission enquiry chatbot. The operational feasibility of a college enquiry chatbot is highly advantageous due to several reasons. Firstly, a chatbot can efficiently handle a large volume of enquiries from prospective students, current students, and other stakeholders. It can provide instant responses and information, eliminating the need for individuals to wait for long periods or navigate through complex phone menus. This enhances user satisfaction and reduces the workload on staff members who would otherwise have to handle these enquiries manually.

Secondly, a chatbot operates 24/7, allowing users to access information at any time, regardless of their location or time zone. This round-the-clock availability ensures that individuals can obtain the information they need conveniently, without being constrained by traditional office hours.

Additionally, a college enquiry chatbot can offer personalized responses tailored to the specific needs of each user. By utilizing natural language processing and machine learning algorithms, the chatbot can analyse the queries and provide accurate and relevant information. This personalization enhances the user experience and helps individuals find the information they are seeking more effectively.

Moreover, a chatbot can integrate with existing systems and databases, allowing it to access and retrieve up-to-date information. For instance, it can provide information about course offerings, admission requirements, application deadlines, and campus facilities. By automating these processes, the chatbot streamlines the information retrieval process, reduces errors, and ensures that users receive accurate and consistent information.

Lastly, implementing a college enquiry chatbot can lead to cost savings for the institution. It eliminates the need for hiring additional staff members solely for handling enquiries, as the chatbot can efficiently handle a significant portion of these interactions. This frees up human resources to focus on more complex and specialized tasks, enhancing overall operational efficiency.

Considering these benefits, the operational feasibility of a college enquiry chatbot is evident. It improves efficiency, accessibility, personalization, and cost-effectiveness, making it a valuable tool for enhancing the overall user experience and streamlining information delivery within a college or university setting.

## **4.3 Economic Feasibility**

The economic feasibility analysis involved evaluating the costs associated with developing, implementing, and maintaining the chatbot system. The economic feasibility of a college enquiry chatbot is significant and can have a positive impact on the institution's finances. Firstly, the implementation of a chatbot reduces the need for additional human resources to handle enquiry-related tasks. This means that the institution can save on hiring and training costs, as well as ongoing salary expenses. By automating the enquiry process, the chatbot can handle a large volume of enquiries simultaneously, resulting in operational cost savings.

Additionally, a chatbot can significantly reduce the response time to enquiries, providing instant and accurate information to users. This can lead to increased user satisfaction and potentially attract more prospective students to the college. With a streamlined enquiry process, there is a higher likelihood of converting enquiries into successful admissions, contributing to the institution's revenue.

Furthermore, a college enquiry chatbot can optimize resource allocation within the institution. By automating routine tasks, staff members are freed up to focus on more value-added activities, such as student counselling, research, and administrative tasks that require human expertise. This utilization of human resources in more strategic roles can lead to improved productivity and efficiency, potentially resulting in cost savings or revenue generation.

Moreover, the implementation and maintenance costs of a chatbot are relatively lower compared to the expenses associated with hiring and training additional staff members. Once the chatbot is developed and deployed, ongoing maintenance and updates are generally more cost-effective than the continuous investment required for human resources.

Overall, the economic feasibility of a college enquiry chatbot lies in its potential to reduce operational costs, increase revenue through improved enquiry conversion rates, optimize resource allocation, and provide a cost-effective solution for handling enquiries. By leveraging automation and artificial intelligence technologies, colleges can achieve financial benefits while enhancing their service quality and responsiveness to prospective and current students.

#### **4.4 Behavioural Feasibility**

The behavioural feasibility of a college enquiry chatbot revolves around the acceptance and adoption of the chatbot by its intended users. It is crucial to assess how users perceive and interact with the chatbot to ensure its effectiveness in meeting their needs.

One aspect of behavioural feasibility is user acceptance. Users must find the chatbot intuitive, user-friendly, and easy to interact with. If users feel comfortable and confident in engaging with the chatbot, they are more likely to embrace it as a useful tool for obtaining information and support. User interface design and user experience play significant roles in shaping the behavioural feasibility of the chatbot.

Additionally, the chatbot's ability to provide accurate and relevant information impacts user trust and confidence. Users must perceive the chatbot as a reliable source of information, and the chatbot's responses should align with their expectations. This requires careful training and continuous learning to improve the chatbot's accuracy and minimize errors.

Personalization is another critical factor in the behavioural feasibility of a college enquiry chatbot. Users often have unique preferences and specific needs. The chatbot should be able to understand and cater to these individual requirements, offering personalized responses and suggestions. By adapting to each user's context and preferences, the chatbot enhances the user experience and increases its behavioural feasibility.

Furthermore, the chatbot's ability to handle complex or sensitive enquiries is essential. Some enquiries may require empathetic and nuanced responses. The chatbot should be programmed to handle such situations appropriately, either by providing appropriate information or directing users to relevant support channels. Ensuring that the chatbot can address a wide range of user queries effectively contributes to its behavioural feasibility.

Lastly, the continuous monitoring and improvement of the chatbot's performance are crucial for behavioural feasibility. User feedback, analytics, and user satisfaction surveys can provide valuable insights into user perceptions and experiences. These inputs can inform necessary adjustments and refinements to the chatbot's behaviour and functionality, ensuring its ongoing acceptability and effectiveness.

The feasibility study concluded that developing a college admission enquiry chatbot is a viable solution. The technical evaluation indicated the availability of suitable NLP frameworks and integration capabilities, ensuring the system's technical feasibility. The operational analysis revealed positive feedback from prospective students, parents, and stakeholders, indicating a high level of user acceptance. The financial assessment demonstrated the potential for cost savings and identified revenue generation opportunities.



## **CHAPTER 5: SYSTEM REQUIREMENTS**

### **5.1 Functional requirements**

- The system must provide clear information about admission policy.
- The system must provide clear and fully detailed information about college.
- The system must provide clear and fully detailed information about colleges' programs.
- The system must provide clear and fully detailed information about colleges' majors.
- The system should clarify the minimum percentage in intermediate for each major.
- The system should clarify the duration of study for each course.
- The system should clarify the parallel study policy for each course.
- The system should provide information about placement policy.
- The system should provide information about hostel facility.

### **5.2 Non-Functional requirements**

- The system shall handle multiple user inputs, if two or more students are chatting with the bot, none of the students has to wait too long to be answered by the system.
- The bot should have a delay in response, to let the student feel like he/she is talking to a human instead of a bot. A little late response from the chatbot makes the student feel as though he is talking to human.
- The system should have the appropriate data set. The correct data is the basis for the chatbot, when the data set is correct and tuned, the chatbot will be trained on it to give the best possible result.
- The system should have a data training. Data training mainly depends on the content targeted at admission and registration deanship.
- The system should prevent abusive language.

## CHAPTER 6: BACKGROUND AND ALTERNATIVES

### 6.1 Chatbot

A chatbot is a computer programme that can mimic a discussion or chat with a user in natural language using messaging applications, websites, or mobile applications. A chatbot is a computer program designed to simulate a conversation with human users via textual or auditory methods. Chatbots are usually powered by artificial intelligence and natural language processing techniques, allowing them to understand and respond to user input in a way that feels human-like. The ultimate goal of a chatbot is to provide a seamless, efficient and personalized user experience. It should be accessible around-the-clock and capable of interacting with people in accordance with their input. As the use of smart devices and IoT technology increased, chatbots were created and quickly gained popularity.

There are two types of chatbots: rule-based and AI-based. Rule-based chatbots are programmed with a specific set of rules and predefined responses. They are limited to the questions and responses they have been programmed to handle and cannot adapt to new situations or requests outside their scope. On the other hand, AI-based chatbots use natural language processing (NLP) and machine learning (ML) algorithms to analyse user input and generate appropriate responses based on their previous conversations.

AI-based chatbots can learn from previous conversations to provide more personalized and contextual responses to users. They can also be programmed to understand user intent, enabling them to anticipate user needs and provide more relevant information. This personalized approach can enhance the user experience and foster customer loyalty.

### 6.2 Types of chatbots

#### a. Base-line chatbot:

Baseline chatbot is a simple, rule-based chatbot that is programmed to respond to specific user inputs with pre-defined messages. It typically uses a set of predefined rules and scripts to handle user queries and provide relevant responses. Baseline chatbots do not use artificial intelligence (AI) or machine learning (ML) algorithms to learn from user interactions. Instead, they rely on a fixed set of responses to handle different user inputs.

Baseline chatbots are often used as a starting point for developing more advanced chatbots. They are relatively easy to build and can be used to provide basic information and support to users. However, they may not be able to handle complex queries or provide personalized responses.

**b. AI chatbot:**

An AI chatbot, also known as an artificial intelligence chatbot or a smart chatbot, is a chatbot that uses natural language processing (NLP) and machine learning (ML) algorithms to understand user inputs and provide relevant responses.

AI chatbots can learn from user interactions and improve their responses over time. They can understand the context of a conversation and provide personalized responses based on user preferences, history, and behaviour.

AI chatbots are often used in customer support, sales, and marketing. They can handle a wide range of queries and provide 24/7 assistance to customers. They can also be integrated with other systems, such as CRM and ERP, to provide a seamless experience for users.

**c. Hybrid model:**

A hybrid chatbot is a chatbot that combines the capabilities of both rule-based and AI-based chatbots. It uses a combination of pre-defined rules and machine learning algorithms to provide responses to user queries.

In a hybrid chatbot, rule-based components are used to handle routine and predictable tasks, while AI-based components are used to handle complex queries and provide personalized responses. This allows for a more efficient and accurate response system, as the chatbot can quickly provide basic information and also handle more complex queries.

A hybrid chatbot can also integrate with other systems and APIs to provide a seamless experience for users. For example, it can access a database of product information to answer questions about specific products or services.

### **6.3 How do chatbot works?**

In a nutshell, and as stated in the definition, people converse with chatbots. Two methods exist for communicating with a chatbot:

**a. Text**

In order to manage the discussion, the chatbot analyses the text input and compares it to prepared information called intents. Even if what the user says spans two or more of these intents, the user's utterance is assigned to one of them. The majority of chatbots will choose the intent with the greatest score and proceed in that direction.

**b. Voice**

Some chatbots may converse with users and comprehend their voice utilising a set of application programming interfaces (APIs) that translate the user's recorded voice into the language of their choice, then translate that voice into words in that language, and finally deal with the translated text as previously described.

Chatbots work through a combination of technologies and algorithms to understand user input and generate appropriate responses. Here's a general overview of how chatbots work:

- **Input Processing:** When a user interacts with a chatbot, the input (text or speech) is received and processed. The chatbot needs to understand the user's intent and extract relevant information from the input.
- **Natural Language Processing (NLP):** NLP is a key component of chatbot technology. It involves breaking down the user input into meaningful components, such as words, phrases, and entities. NLP techniques help the chatbot understand the context, identify the intent behind the user's message, and extract important details.
- **Intent Recognition:** The chatbot's AI system tries to determine the user's intent based on the input received. This involves mapping the user's message to predefined intents or categories. For example, if a user asks about the weather, the intent might be "Get Weather Information."
- **Entity Extraction:** Along with intent recognition, the chatbot also identifies specific entities or parameters from the user input. Entities are pieces of information that the chatbot needs to understand to provide an accurate response. For example, in a weather query, the entities could be the location or date.
- **Dialog Management:** Once the intent and entities are identified, the chatbot uses dialog management techniques to maintain the context of the conversation. Dialog management allows the chatbot to keep track of previous interactions, handle multi-turn conversations, and manage the flow of the conversation.
- **Response Generation:** Based on the user's intent and extracted entities, the chatbot generates a response. This can be done through predefined responses for rule-based chatbots or through machine learning algorithms for AI-powered chatbots. The response can be a simple text message or a more complex action, such as retrieving information from a database or performing a task.
- **Natural Language Generation (NLG):** NLG is the process of converting the chatbot's generated response into human-readable text. It ensures that the chatbot's response is grammatically correct, coherent, and appropriate for the conversation.
- **Output Presentation:** The chatbot presents the response to the user, either as a text message or as speech output. In text-based chatbots, the response is typically displayed on the user interface, such as a messaging app or website. In voice-based chatbots, the response is converted into speech and delivered to the user through text-to-speech synthesis.
- **Learning and Improvement:** Chatbots can continuously learn and improve over time. They can be trained on large datasets or through user interactions to enhance their understanding of user inputs and generate more accurate responses. Machine learning algorithms are often used to improve the chatbot's performance.

It is important to note that the complexity of a chatbot's workings can vary depending on the specific implementation and technologies used. Rule-based chatbots follow predefined rules and decision trees, while AI-powered chatbots utilize machine learning models and algorithms to learn and adapt to user interactions.

Overall, chatbots rely on the integration of NLP, machine learning, and dialog management techniques to understand user input, determine intent, generate responses, and provide an interactive and conversational experience.

## **6.4 Options to build a chatbot**

### **a. From scratch**

To ensure efficiency and accuracy, we must first decide the opportunities for our chatbot and its area and scope. and in order to address the operational difficulties, a precise grasp of the client needs is needed. The user's interaction with your app or website will thereafter be significantly influenced by the bot's design. and chatbot conversations can be divided into organised and unstructured interactions.

- Structured interaction. You are already familiar with this type of conversation. It's similar to a FAQ area of your app or website because you can quickly construct it and anticipate the questions your consumers will ask. Your contact details, services, products, and other information will be linked to this information.
- Unstructured interaction. The free-form plain text is part of the unstructured discussion flow. It's difficult to foresee what questions will come up, and it seems like your chatbot is competing in an extempore speech competition. Here, AI plays a key role; via NLP analysis, it decodes the text's context. whereas the chatbot will receive a voice from the same NLP.

The later choice will need specialized chatbot developers with an understanding of programming languages, machine learning, and AI. We can use some of the code-based frameworks to build and handle the chatbot like wit.ai and api.ai.

### **b. Using platforms**

It is similar to scratch chatbots but the only difference is that you do not have to hire a specialized developer and use the chatbot builder platforms like Chatfuel, Botsify and Rasa, it's not hard or impossible to achieve it. but it's not possible to create an NLP-enabled chatbot that can deal with unstructured data.

If you don't have extensive programming knowledge, you can use chatbot builders or no-code platforms. These platforms allow you to create chatbots using a visual interface and pre-built templates. They often provide drag-and-drop functionality for designing conversation flows, and you can customize responses and integrate with external services. Some popular chatbot builder platforms include Chatfuel, ManyChat, and Tars.

### c. Open-Source Libraries

There are open-source libraries and frameworks available that provide building blocks for chatbot development. For instance, you can use the open-source library ChatterBot in Python, which offers pre-trained language models and a simple API to build conversational agents. Other options include Botpress, an open-source chatbot platform, and Rasa, an open-source framework for building AI-powered chatbots.

### d. Natural Language Understanding (NLU) Services

Another option is to use natural language understanding (NLU) services provided by major tech companies. These services offer pre-trained models and APIs that handle the NLP and intent recognition tasks. Examples include Google Cloud's Dialogflow, Amazon Lex, and IBM Watson NLU. You can integrate these services into your application and focus on building the conversation flow and response generation.

### e. Hybrid approaches

You can also combine multiple approaches based on your specific requirements. For example, you can use a chatbot development framework for intent recognition and dialog management but implement custom response generation based on your needs. This approach allows for greater flexibility and customization.

## 6.5 Chatbot Platforms alternatives:

- a. **IBM Watson:** is promoted as a question-and-answer platform that may be used to create chatbots and applications. The IBM Watson platform enables us to develop a tool that shares a dialogue exchange between users of Quick n' Easy Projector Rentals and our chatbot. The IBM interface is easy to use, and at first look no back-end coding is visible. The chatbot is simple to include into other programmes like Slack, Facebook, and Twilio.
- b. **Google Dialogflow:** is a conversational agent that is simple to understand. Theoretically, by comprehending intents, entities, and dialogue control, we can have a bot up and running. As previously said, the majority of the chatbot systems we tested adhere to these general characteristics.
- c. **Rasa:** is a natural language processing tool-equipped open source chatbot. The free software is known as Rasa NLU. Rasa's machine learning algorithm can be altered and customised to produce a model that yields the desired outcomes. Rasa NLU may be used wherever you choose, and you don't have to give your chatbot training data to Google, Microsoft, Amazon, or Facebook to do it.

## 6.6 Selected Platform:

We have used Rasa framework to build the required chatbot. Rasa is a powerful tool for building chatbots and voice assistants, but why did we choose rasa?

Rasa has a lot of advantages such as:

- Customizable: Rasa is highly customizable and flexible, allowing developers to build chatbots and voice assistants that meet their specific needs and requirements.
- Open-Source: Rasa is an open-source framework, meaning that it's free to use and developers can contribute to its development and improvement.
- Multilingual: Rasa supports multiple languages, making it easier to build chatbots and voice assistants for users who speak different languages.
- Integration: Rasa can be integrated with other systems, such as customer relationship management (CRM) tools, to improve the chatbot's functionality and effectiveness.
- Machine Learning: Rasa's machine learning algorithms enable it to learn from user interactions and improve its performance over time.
- Contextual Awareness: Rasa's natural language processing (NLP) engine can understand the context of user queries, allowing it to provide more accurate and relevant responses.
- Dialog Management: Rasa's built-in dialog management tools enable developers to create complex conversational flows and handle user interactions more effectively.
- Community Support: Rasa has a large and active community of developers and users, providing access to resources, tutorials, and support.

But as we know, nothing is perfect and rasa has its disadvantages like:

- Steep learning curve: Rasa requires significant technical expertise and programming knowledge to use effectively. This can make it difficult for non-technical users to get started and require more time and resources to train developers.
- Limited pre-built functionality: Unlike some other chatbot development frameworks, Rasa does not come with pre-built components or templates for common features. This can require developers to build everything from scratch, which can be time-consuming and resource-intensive.
- Dependency management: Rasa has a large number of dependencies, which can be challenging to manage and update. This can result in compatibility issues and make it difficult to keep up with the latest updates and security patches.
- Scalability: While Rasa is designed to be scalable, it can be challenging to scale up as your chatbot grows in complexity. This can require significant effort and resources to maintain performance and stability.

- Natural Language Understanding (NLU) accuracy: Rasa's NLU component relies on machine learning algorithms, which can be less accurate than traditional rule-based systems. This can result in issues with understanding user input, especially for more complex or nuanced queries.

## 6.7 How to Setup RASA and start a chatbot

- First, we have to know that Rasa needs python to work, so we'll install Python and the version should be from 3.7 to 3.10.



Fig.-6.1: Python download screen

Check if your Python environment is already configured by opening the cmd: type `python3 --version`

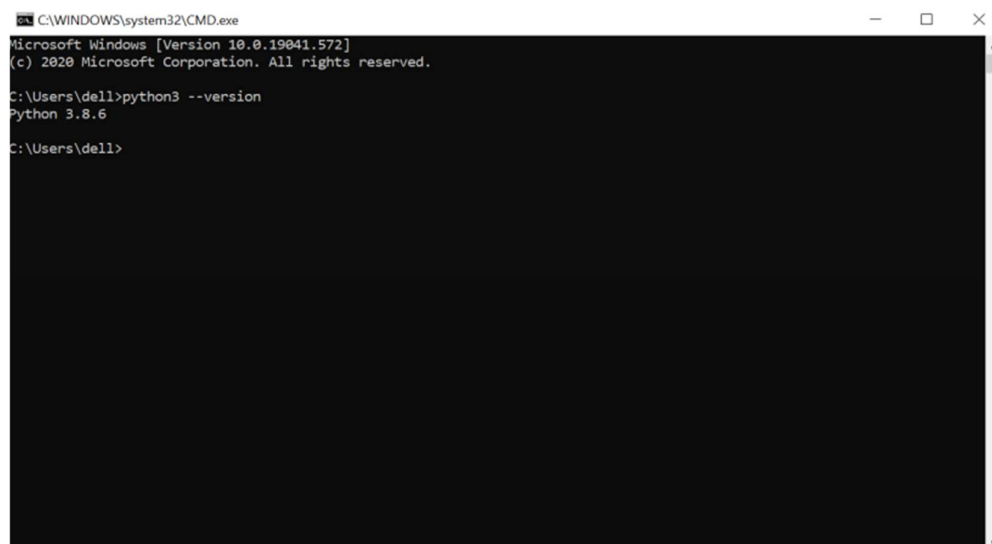


Fig. 6.2: Testing python version



If you did NOT find a version, then type this command: `pip3 install -U pip`

- Virtual Environment Setup

Create and activate the virtual environment using the below commands.

```
python3 -m venv ./venv
```

Activate the virtual environment:

```
./venv\Scripts\activate
```

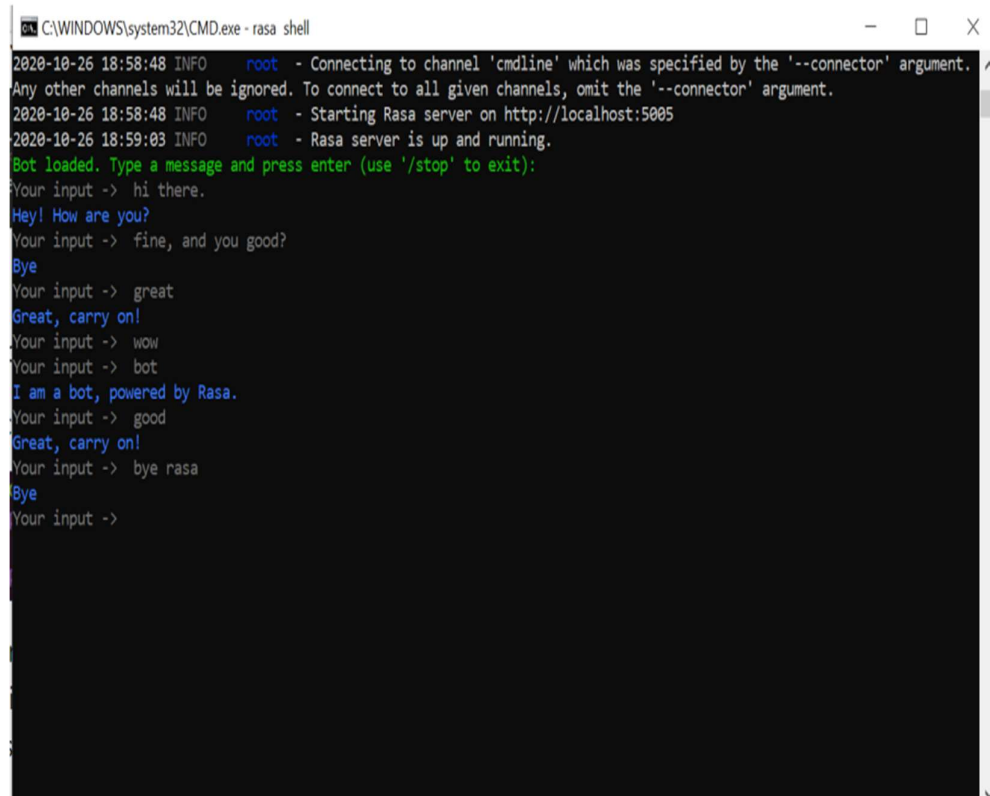
- Install Rasa open-source with the below command:

```
pip install rasa
```

- make a directory and move to it

```
mkdir test-chatbot && cd test-chatbot
```

Create the new project with **rasa init** command and start the conversation with the initial demo chatbot.



```
C:\WINDOWS\system32\CMD.exe - rasa shell
2020-10-26 18:58:48 INFO     root - Connecting to channel 'cmdline' which was specified by the '--connector' argument.
Any other channels will be ignored. To connect to all given channels, omit the '--connector' argument.
2020-10-26 18:58:48 INFO     root - Starting Rasa server on http://localhost:5005
2020-10-26 18:59:03 INFO     root - Rasa server is up and running.
Bot loaded. Type a message and press enter (use '/stop' to exit):
Your input -> hi there.
Hey! How are you?
Your input -> fine, and you good?
Bye
Your input -> great
Great, carry on!
Your input -> wow
Your input -> bot
I am a bot, powered by Rasa.
Your input -> good
Great, carry on!
Your input -> bye rasa
Bye
Your input ->
```

Fig. 6.3: Rasa Bot initial chatting

## CHAPTER 7: MODELS AND INTERFACES

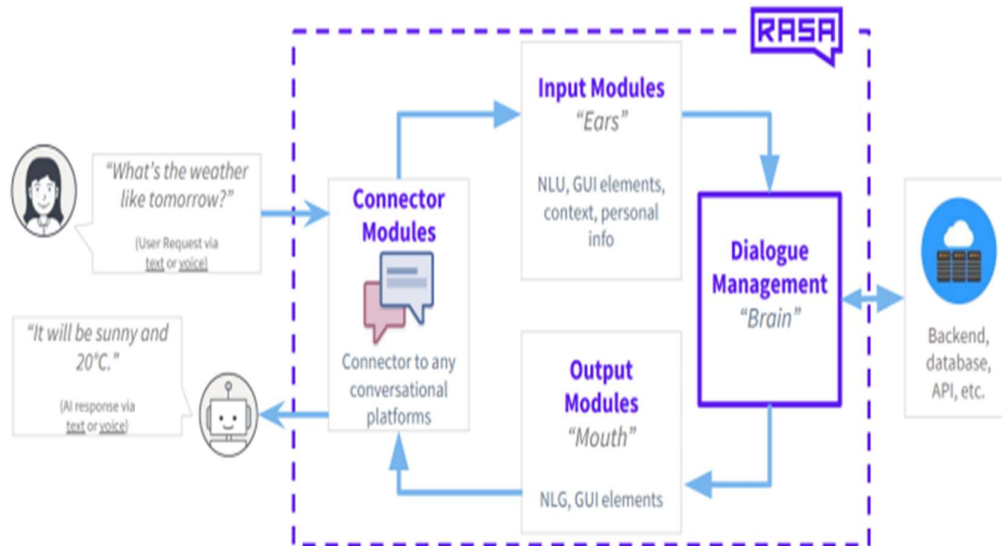
### 7.1 Basics of Rasa Opensource Conversational AI:

Rasa is an open-source framework that allows you to build and deploy conversational AI chatbots and virtual assistants. It provides a set of tools and libraries that enable developers to create sophisticated and interactive conversational experiences.

Here are the basic components and concepts in Rasa:

- 1 NLU (Natural Language Understanding): Rasa's NLU component is responsible for understanding and extracting the user's intent and entities from the input text. It uses machine learning algorithms to classify user messages into intents and extract relevant information as entities.
- 2 Core: Rasa Core handles the dialogue management of the chatbot. It uses machine learning techniques, specifically reinforcement learning, to learn and predict the next best action based on the current conversation state.
- 3 Stories: Stories are example conversations that are used to train the dialogue model in Rasa Core. They define a sequence of user inputs and expected bot responses, along with corresponding intents and entities.
- 4 Actions: Actions represent the behaviour or responses of the bot. Actions can include sending a message, making an API call, querying a database, or any other custom behaviour. Rasa allows you to define custom actions that can be triggered based on the predicted action from the dialogue model.
- 5 Training Data: To train the NLU and Core models, you need training data. Training data for NLU consists of user messages along with their corresponding intents and entities. Training data for Core consists of stories that represent example conversations.
- 6 Domain: The domain file in Rasa defines the intents, entities, actions, and templates used by the bot. It also includes the configuration for the NLU and Core models.
- 7 Pipelines: Rasa allows you to configure the NLU pipeline, which defines the sequence of components used for intent classification and entity extraction. You can choose from various pre-built components or create custom ones.
- 8 Training and Evaluation: Rasa provides a command-line interface to train and evaluate your models. You can use the training data to train the NLU and Core models and evaluate their performance.
- 9 Integration: Rasa can be integrated with various messaging platforms, such as Slack, Facebook Messenger, or custom chat interfaces. It provides connectors and adapters to handle communication between the chatbot and the messaging platform.

The diagram below shows the general rasa open source conversational associated with machine learning.



**Fig. 7.1 Basics of Rasa Opensource Conversational AI**

## 7.2 Architectural Model

"The Rasa Opensource architecture is depicted in the diagram below. Natural Language Understanding (NLU) and dialogue management are the two main parts. Entity extraction, intent classification, and answer retrieval are all handled by the NLU component. As evidenced, because it processes user utterances utilising a generated NLU model, it is referred to as the NLU Pipeline below utilising the trained pipeline. The dialogue management element selects the subsequent course of action in a discussion tailored to the situation. In the diagram, this is represented by the dialogue policies.

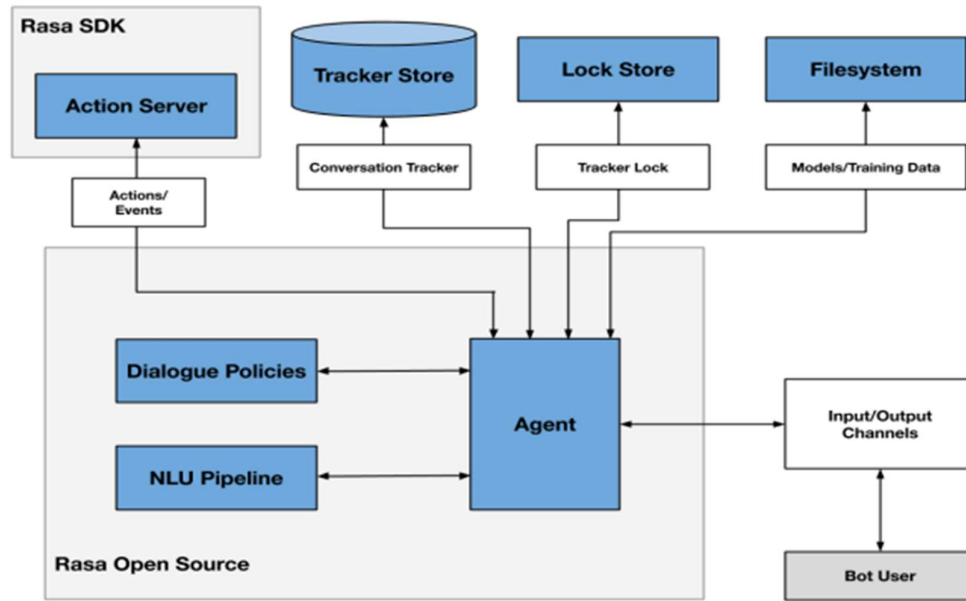


Fig. 7.2: Architectural Model

1. **NLU (Natural Language Understanding) Module:** This module is responsible for processing user input and extracting relevant information from it, such as intent and entities. It uses machine learning models to analyse user input and classify it into meaningful categories.
2. **Dialogue Management:** This module is responsible for managing the conversation flow between the user and the chatbot. It uses a state machine to keep track of the current conversation state and determine the appropriate response to user input.
3. **Action Server:** This module is responsible for executing actions in response to user input. It can be used to access external APIs or databases, perform calculations, or send messages to users.
4. **Core:** This module is responsible for managing the overall behaviour of the chatbot. It uses a reinforcement learning algorithm to learn from user interactions and improve its performance over time.
5. **Tracker:** This module is responsible for tracking the conversation history and storing user data. It maintains a history of user input, chatbot responses, and conversation context.
6. **APIs:** Rasa provides APIs that can be used to integrate with external systems, such as CRM systems or messaging platforms.

Overall, the Rasa architecture is designed to be flexible and modular, allowing developers to customize their chatbot solutions based on their specific requirements. Its use of machine learning algorithms and reinforcement learning allows for intelligent conversation management and continuous improvement over time.

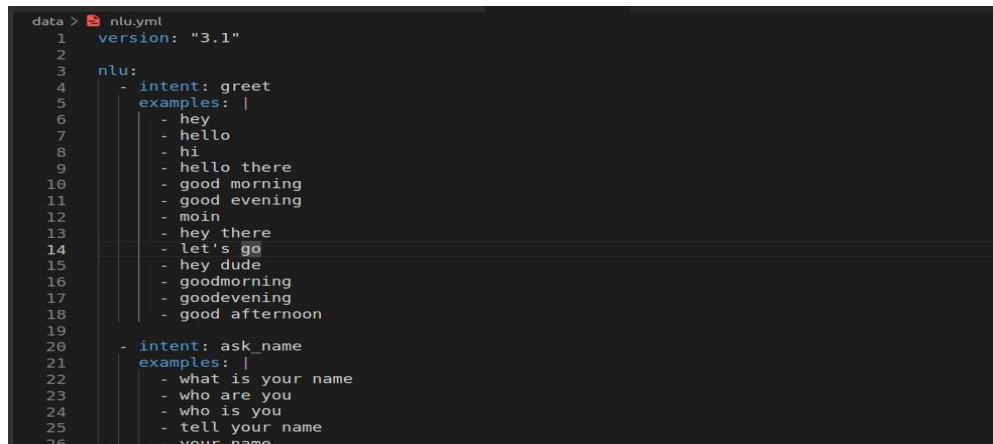
### 7.3 Rasa framework files

Dealing with the framework changed and became better by using files with .yaml extensions.

- **nlu.yaml:**

The file contains the Natural Language Understanding (NLU) model training examples. This includes intents, which are user goals, and example utterances that represent those intents. The NLU training data also labels the entities, or important keywords, the assistant should extract from the example utterance.

- **Entities:** The important keywords that an assistant should take note of. For example, the message 'My name is Maher' has the name 'Maher' in it. An assistant should extract the name and remember it throughout the conversation to keep the interaction natural.
- **Utterances:** Anything a user says. A single utterance is an entire sentence passed as input to the Chatbot to intent. By definition, an utterance holds an intent and could potentially include one or more entities. During each transaction with the Chatbot, an utterance is sent to the Chatbot's understanding unit to parse and interpret.



```
data > nlu.yaml
1  version: "3.1"
2
3  nlu:
4    - intent: greet
5      examples: |
6        - hey
7        - hello
8        - hi
9        - hello there
10       - good morning
11       - good evening
12       - moin
13       - hey there
14       - let's go
15       - hey dude
16       - goodmorning
17       - goodevening
18       - good afternoon
19
20    - intent: ask_name
21      examples: |
22        - what is your name
23        - who are you
24        - who is you
25        - tell your name
26        - your name
```

Fig 7.3: nlu.yaml file content

- **domain.yaml:** Defines the environment in which the assistant operates, including:
  - What the user means: specifically, what intents and entities the model can understand.
  - What responses the model can provide: such as utterances or custom actions.
  - What to say next: what the model should be ready to respond with.
  - What info to remember: what info an assistant should remember and use throughout the conversation.

```

domain.yml
1 version: "3.1"
2
3 intents:
4   - greet
5   - goodbye
6   - fee_structure
7   - fee_misc
8   - admission_process
9   - hod
10  - placement
11  - random
12  - ask_name
13
14 slots:
15   course:
16     type: text
17     influence_conversation: true
18     mappings:
19       - type: from_entity
20         entity: course
21
22 entities:
23   - course
24
25 actions:
26   - action_utter_fees

```

```

data > nlu.yml
1 version: "3.1"
2
3 nlu:
4   - intent: greet
5     examples: |
6       - hey
7       - hello
8       - hi
9       - hello there
10      - good morning
11      - good evening
12      - moin
13      - hey there
14      # - let's go
15      - hey dude
16      - goodmorning
17      - goodevening
18      - good afternoon
19
20   - intent: ask_name
21     examples: |
22       - what is your name
23       - who are you
24       - who is you
25       - tell your name
26       - your name

```

Fig. 7.4 domain.yml and nlu.yml files.

- **data/stories.yml:** example end-to-end conversations. A transcription of a conversation between a user and a Chatbot, where user inputs are expressed as corresponding intents (and entities where necessary), and the Chatbot's responses are expressed as corresponding action names.

```

domain.yml
1 version: "3.1"
2
3 intents:
4   - greet
5   - goodbye
6   - fee_structure
7   - fee_misc
8   - admission_process
9   - hod
10  - placement
11  - random
12  - ask_name
13
14 slots:
15   course:
16     type: text
17     influence_conversation: true
18     mappings:
19       - type: from_entity
20         entity: course
21
22 entities:
23   - course
24
25 actions:
26   - action_utter_fees

```

```

data > stories.yml
1 version: "3.1"
2
3 stories:
4   - story: happy path
5     steps:
6       - intent: greet
7       - action: utter_greet
8       - action: utter_help
9
10  - story: ask name
11    steps:
12      - intent: ask_name
13      - action: utter_name
14
15  - story: fees enquiry
16    steps:
17      - intent: fee_structure
18      - action: action_utter_fees
19
20  - story: miscellaneous fees enquiry details
21    steps:
22      - intent: fee_misc
23      - action: utter_fees_misc
24
25  - story: fees enquiry with greet
26    steps:

```

Fig. 7.5: domain and stories files

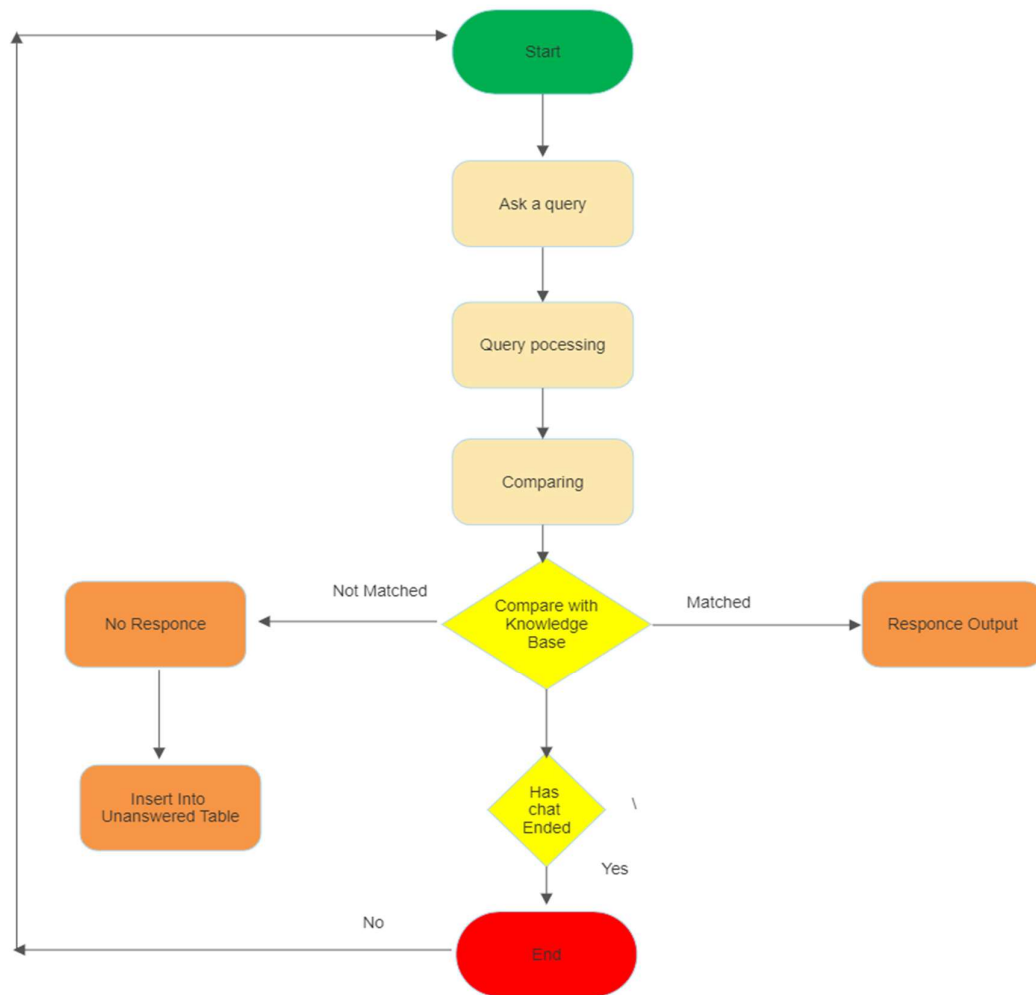
- **actions/actions.py:** all programming is here. The action class describes how the Chatbot fulfils various intents. Each action has access to the tracker, which contains data about the conversation, and implements a run function which holds the logic for carrying out said action.

- **Tracker:** A tracker object maintains the current state of the conversations. It keeps track of the events that have happened so far, such as utterances and actions, as well as other data such as the slots and entities.
- **Slots:** Slots are variables a Chatbot requires to perform a specific task. Slots are essential to interpret a user's input and adequately execute the action. Slots are commonly filled using entities. Slots serve as the building block for a Chatbot's context manager.
- **Processing pipeline/config.yml:** The configuration file defines the components and policies that the model will use to make predictions based on user input.
- **Endpoints.yml:** A web server that reacts to the call, runs the code, and optionally, returns information to modify the dialogue state. The full configuration of the custom action server is provided in this file.
- **Credentials.yml:** This file holds the configuration details for connecting to external applications, like messaging channels.

## 7.4 Flowchart and Sequence Diagram

### Flowchart

1. Start: The chatbot starts.
2. User Input: The chatbot waits for the user to input a message or question.
3. Process Input: The chatbot processes the user's input, which involves understanding the intent and extracting relevant information.
4. Generate Response: Based on the processed input, the chatbot generates an appropriate response.
5. Send Response: The chatbot sends the generated response back to the user.
6. End: The chatbot reaches the end of the flowchart, and the conversation may continue with the user's next input.



**Fig. 7.6: Flowchart of chatbot**

### Sequence Diagram

The figures given below illustrate the Sequence Diagram of the chatbot. A sequence diagram simply displays the order in which things interact, or the order in which these interactions occur. Sequence diagrams show how and in what sequence the components of a system work together system work together. Businesspeople and software engineers frequently use these diagrams to describe and understand requirements for new and current systems.



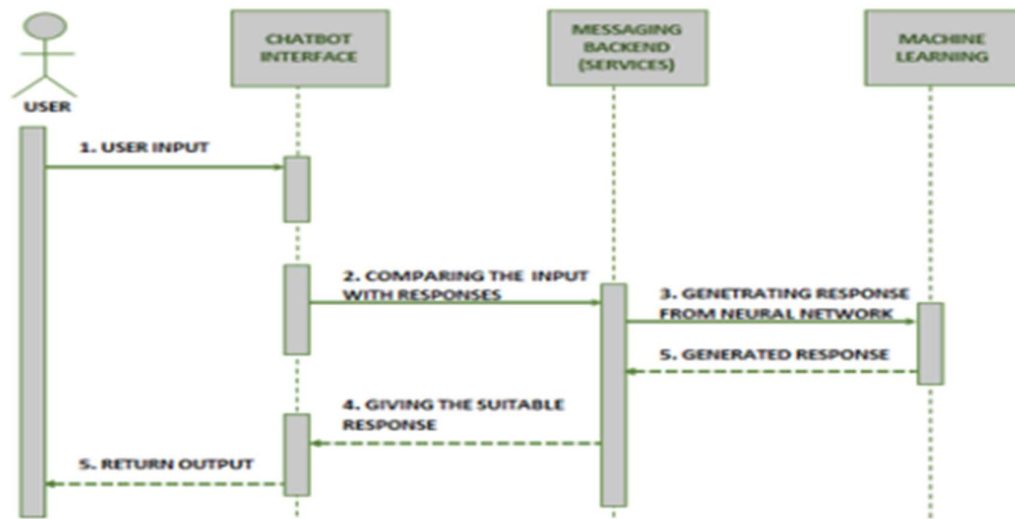


Fig. 7.7: Sequence diagram of user

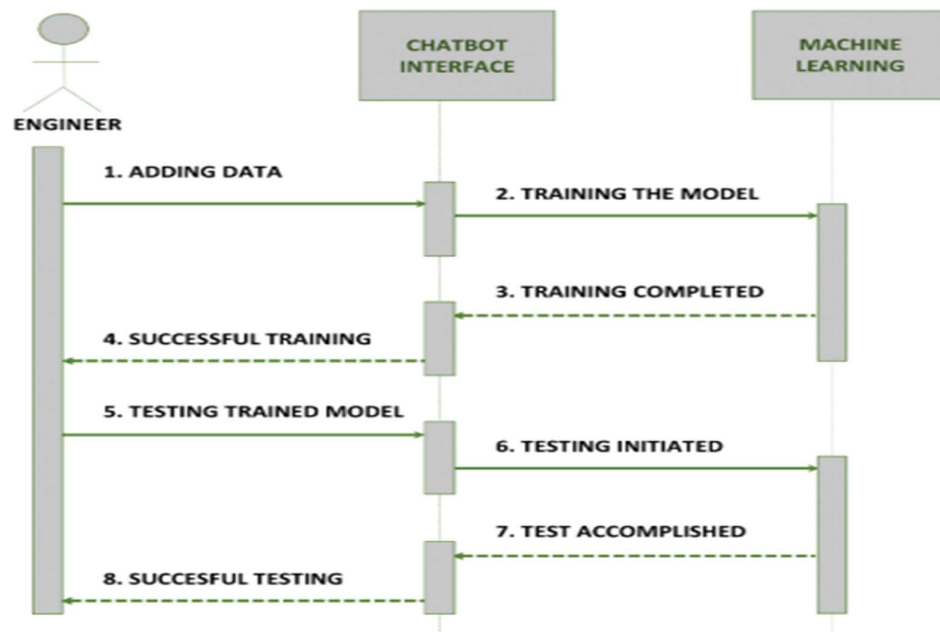


Fig. 7.8: Sequence diagram of developer

## CHAPTER 8: SYSTEM DEMONSTRATION

### 8.1 Implementation details

The project consists of several parts, the most important of which are:

- 1 Actions directory is the folder that contains 3 files in the Python programming language. The most important file is the Actions file, which contains custom Actions that are built according to the need and purpose of the chatbot.
- 2 The data section and contains 3 important files that cannot be dispensed with: the natural language understanding file, which contains the training data necessary for the bot, which it is expected to receive during its operation from the user, and the rules file, which contains a certain structure that makes the bot act obligatory according to what exists, regardless of the circumstances in terms of the received data, and the story file, which contains scenarios of conversations with users, and all conversations are recorded within this file in .yaml extension.
- 3 There is also a very important file, which is the domain that defines the universe in which your assistant operates. It specifies the intents, entities, slots, responses, forms, and actions your bot should know about. It also defines a configuration for conversation sessions.
- 4 There are also configuration files, endpoints, and credentials that are responsible for the overall properties and some permissions allowed for the bot and linking with chat channels such as Slack and Facebook Messenger.
- 5 There is also the Models section, in which all models are stored after each bot training  
Every model we can say is like the nucleus or brain of the bot. The bot cannot work and listen to the user's messages and respond to them without the model.
- 6 After each bot training process, and to get the latest results, you must choose the newest model. Older models can be selected so that they can be compared with the new model in terms of additions.

### 8.2 Implementation issues

We faced many problems. In each stage we passed, we encountered many problems, some of which took days to solve. We will mention the most important problems that we encountered and stood in our way:

1. The problem of installing the framework, during the installation on Windows we did not encounter any serious problems except for the problem of installing Rasa X, which is the GUI. The last version was not very compatible with Windows at the time, so we had to use one of the old versions so that we could work on Rasa X.
2. Installing the framework on Linux was one of the biggest problems we faced, and the problem was divided into smaller problems: the issue of the Python programming language version, the problem of pip packages, and the problem of environment variables and dependencies.

3. We also encountered problems with the actions, especially that the file was sensitive to the issue of functions' names and how to deal with it and not respond except by relying on the correct and acceptable naming rules.
4. The biggest problem we faced was the problem of integrity error while working with Rasa X and its connection with the Action Server on Windows. The problem revolved around the inability to record or respond to commands and conversations between the user and the bot in the dedicated database. We repeatedly searched, asked, and investigated this problem that has not been resolved at the time of writing this report, as it was a problem related to the Windows operating system. We did not encounter the problem while running the bot on Linux.
5. One of the problems that we faced was about slots and not filling them out according to the rules in the Domain file. While talking to the bot, some answers are extracted and processed, and responded to according to what was extracted and we encountered the problem on Windows.
6. The bot training process is one of the most frequent problems in terms of time and space, so every training process for the bot requires at least 4 minutes at least. After completing the training process, a model with a size of more than 20 MB will be produced. And you must activate the latest model after each training process.
7. Not all buttons appear when the bot is connected to chat channels such as Slack and Facebook Messenger.
8. Uploading the bot to Docker was one of the most difficult problems for us due to our lack of knowledge of Docker and the mechanism of uploading projects to it and the necessity of using Linux.

### **8.3 Important information**

When training the bot, it is important to make sure that the structure in the domain file is in the right format or it will fail.

- Training time is machine dependent.
- It is ok to delete old models, but it is not recommended.
- Intents in the NLU file have to contain at least two training examples. One training example may lead to training failures.
- Intents have to be part of the domain. If not, it may lead to issues in the conversations.
- Any change in the actions file requires to stop the action server and restart it again by typing `rasa run actions` from the terminal ONLY to acquire the changes.
- Any change in `nlu`, `rules`, `stories`, `config`, `credentials`, `domain`, and `endpoints` files requires to stop the bot and restart it again in case of using terminal. In case of using Rasa X GUI, the bot won't stop or need to be restarted.
- In case Rasa X was stopped, the ONLY way to start it again is using the terminal by typing `rasa x`.
- User needs a password to have access to the internal Rasa X GUI, it is automatically generated in every attempt to start Rasa X from the terminal.

- Rasa X password is not changed when attempting to train the bot using the GUI.
- It's ok to have a User Warning about the model confidence being set to softmax instead of linear norm. Rasa framework developers recommend the softmax instead of the linear norm.

## 8.4 Snapshots

- **Rasa training log screen**

The Rasa training log screen provides valuable insights into the training process and performance of your chatbot's machine learning models. It displays detailed information about the training data, model architecture, and training metrics. Here is an explanation of the key components you'll typically find on the Rasa training log screen:

**Epochs:** The training process in Rasa involves iterating over multiple epochs. An epoch represents a complete pass through the training data. The log screen usually shows the progress of each epoch, indicating how many iterations have been completed.

**Training Data:** The log screen displays information about the training data used for model training. This includes the number of training examples, stories, and dialogue turns. It provides a summary of the data used to train the NLU (Natural Language Understanding) and Core (dialogue management) models.

**Model Architecture:** The log screen shows the details of the model architecture being trained. It specifies the configuration and layers of the neural network models used for NLU and Core training. This information helps you understand the structure of the underlying models.

**Training Metrics:** The log screen provides various training metrics that give insights into the performance of the models during training. These metrics help you evaluate the quality of the trained models and identify potential areas for improvement. Common training metrics include loss values, accuracy, F1 score, and validation scores.

**Training Progress:** The log screen displays a progress bar or status indicators to show the training progress of each epoch. It gives you an idea of how far the training process has progressed and how many iterations are remaining.

**Training Time:** The log screen often includes the duration or elapsed time of the training process. It gives you an estimate of how long the training took to complete, allowing you to assess the training efficiency.

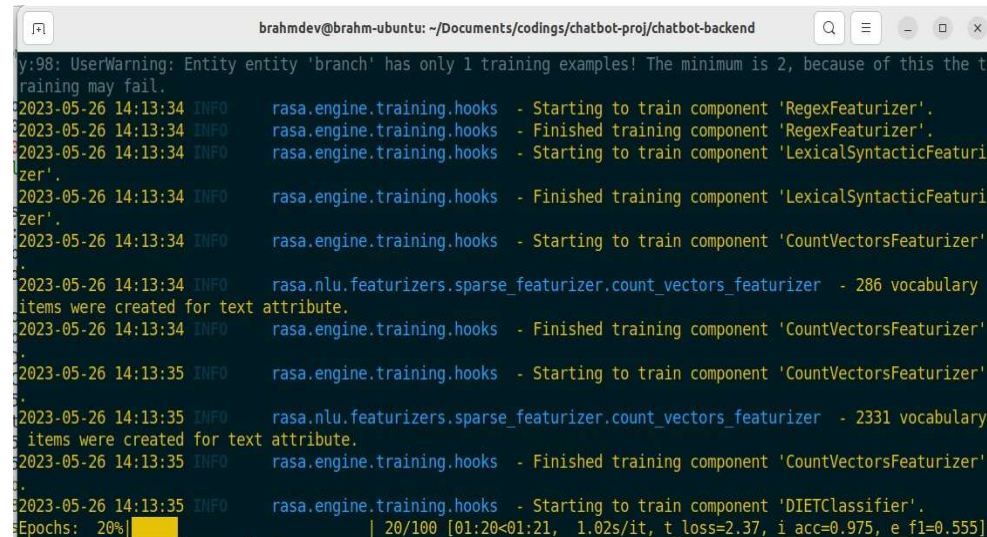
**Log Messages:** During the training process, Rasa outputs informative log messages that provide details about the training steps, data processing, and model performance. These messages help you monitor the training process and identify any potential issues or warnings.

By observing the training log screen, you can gain insights into the training progress, model performance, and potential areas for improvement. It helps you evaluate the effectiveness of the training data, assess the impact of training

hyperparameters, and make informed decisions regarding model adjustments or data augmentation techniques.

It is important to note that the specific format and details of the training log screen may vary depending on the version of Rasa and the training configuration you're using. However, the general purpose of providing information about the training process and model performance remains consistent across versions.

The command `rasa train` trains the bot with the specified questions in the `nlu.yml` files. It trains the whole nlu model and saves it in `models` directory.



```
brahmdev@brahm-ubuntu: ~/Documents/codings/chatbot-proj/chatbot-backend
y:98: UserWarning: Entity entity 'branch' has only 1 training examples! The minimum is 2, because of this the t
raining may fail.
2023-05-26 14:13:34 INFO rasa.engine.training.hooks - Starting to train component 'RegexFeaturizer'.
2023-05-26 14:13:34 INFO rasa.engine.training.hooks - Finished training component 'RegexFeaturizer'.
2023-05-26 14:13:34 INFO rasa.engine.training.hooks - Starting to train component 'LexicalSyntacticFeaturi
zer'.
2023-05-26 14:13:34 INFO rasa.engine.training.hooks - Finished training component 'LexicalSyntacticFeaturi
zer'.
2023-05-26 14:13:34 INFO rasa.engine.training.hooks - Starting to train component 'CountVectorsFeaturizer'
2023-05-26 14:13:34 INFO rasa.nlu.featurizers.sparse_featurizer.count_vectors_featurizer - 286 vocabulary
items were created for text attribute.
2023-05-26 14:13:34 INFO rasa.engine.training.hooks - Finished training component 'CountVectorsFeaturizer'
2023-05-26 14:13:35 INFO rasa.engine.training.hooks - Starting to train component 'CountVectorsFeaturizer'
2023-05-26 14:13:35 INFO rasa.nlu.featurizers.sparse_featurizer.count_vectors_featurizer - 2331 vocabulary
items were created for text attribute.
2023-05-26 14:13:35 INFO rasa.engine.training.hooks - Finished training component 'CountVectorsFeaturizer'
2023-05-26 14:13:35 INFO rasa.engine.training.hooks - Starting to train component 'DIETClassifier'.
Epochs: 20%|██████| 20/100 [01:20<01:21, 1.02s/it, t_loss=2.37, i_acc=0.975, e_f1=0.555]
```

Fig. 8.1 Rasa training log screen

- **Rasa action server terminal screen**

The Rasa Action Server Terminal Screen is a console or command line interface that displays information and logs related to the Rasa Action Server. The Action Server is responsible for handling custom actions in your Rasa chatbot, such as executing business logic, making API calls, or accessing external resources.

Here is an explanation of the key components you may find on the Rasa Action Server Terminal Screen:

**Startup Information:** When you start the Rasa Action Server, the terminal screen typically displays information about the server's startup process. This includes details such as the server address, port number, and any other configurations you have specified.

**Requests and Responses:** As the action server runs, it logs information about incoming requests and outgoing responses. It shows details of each request received from the Rasa Core, including the intent, entities, and any other relevant information. Similarly, it logs the actions taken by the server and the responses it sends back to the Rasa Core.

**Custom Action Execution:** The terminal screen provides logs related to the execution of custom actions defined in your chatbot. It shows the sequence of actions being executed, along with any relevant information or data associated with each action. This helps you track the flow and execution of your custom actions.

**Error Messages and Debugging:** If there are any errors or exceptions encountered during the action server's operation, the terminal screen displays error messages and traceback information. These messages help you identify and debug issues in your custom actions or server setup.

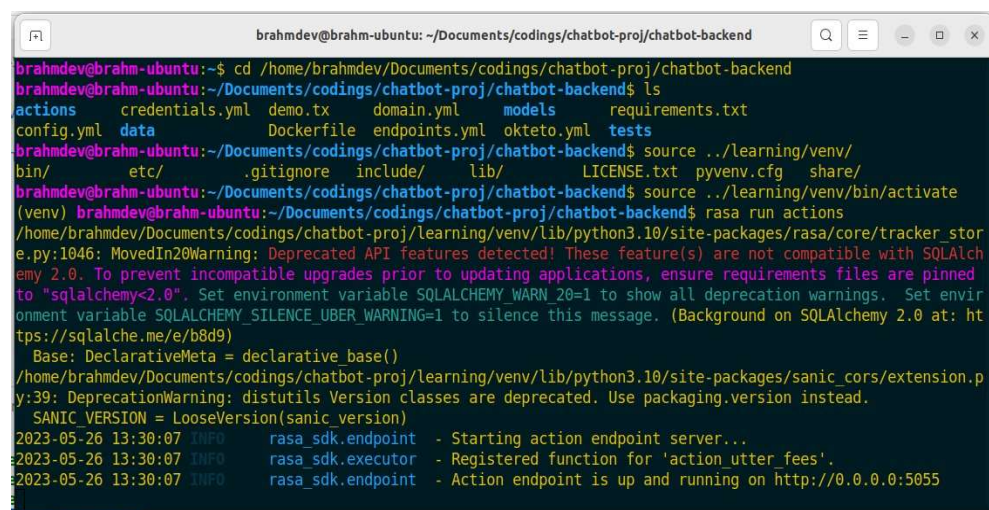
**Server Status and Health Checks:** The terminal screen may include status information about the action server, such as its health or connectivity to external resources. It can display logs related to any health checks being performed or connection attempts made by the server.

**Server Interactions and External APIs:** If your custom actions involve making API calls or interacting with external services, the terminal screen can show logs related to those interactions. It may display details of the API requests sent by the action server and the corresponding responses received from the external services.

By monitoring the Rasa Action Server Terminal Screen, you can observe the execution of your custom actions, track requests and responses, debug any errors, and gain insights into the server's overall operation.

It is important to note that the specific format and details of the terminal screen may vary depending on your operating system, terminal configuration, and the version of Rasa you are using. The logs displayed on the terminal screen can provide valuable information for troubleshooting and understanding the behaviour of your Rasa Action Server.

Rasa action server runs custom actions for a Rasa Opensource conversational assistant. The command *rasa run actions* starts the custom action server with the actions classes defined in the *actions/actions.py* file.



```
brahmdev@brahm-ubuntu: ~/Documents/codings/chatbot-proj/chatbot-backend
brahmdev@brahm-ubuntu:~/Documents/codings/chatbot-proj/chatbot-backend$ ls
actions  credentials.yml  demo.tx  domain.yml  models  requirements.txt
config.yml  data  Dockerfile  endpoints.yml  okteto.yml  tests
brahmdev@brahm-ubuntu:~/Documents/codings/chatbot-proj/chatbot-backend$ source ../learning/venv/bin/activate
(venv) brahmdev@brahm-ubuntu:~/Documents/codings/chatbot-proj/chatbot-backend$ rasa run actions
/home/brahmdev/Documents/codings/chatbot-proj/learning/venv/lib/python3.10/site-packages/rasa/core/tracker_store.py:1046: MovedIn20Warning: Deprecated API features detected! These feature(s) are not compatible with SQLAlchemy 2.0. To prevent incompatible upgrades prior to updating applications, ensure requirements files are pinned to "sqlalchemy<2.0". Set environment variable SQLALCHEMY_WARN_20=1 to show all deprecation warnings. Set environment variable SQLALCHEMY_SILENCE_UBER_WARNING=1 to silence this message. (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
Base: DeclarativeMeta = declarative_base()
/home/brahmdev/Documents/codings/chatbot-proj/learning/venv/lib/python3.10/site-packages/sanic_cors/extension.py:39: DeprecationWarning: distutils Version classes are deprecated. Use packaging.version instead.
SANIC_VERSION = LooseVersion(sanic_version)
2023-05-26 13:30:07 INFO rasa_sdk.endpoint - Starting action endpoint server...
2023-05-26 13:30:07 INFO rasa_sdk.executor - Registered function for 'action_utter_fees'.
2023-05-26 13:30:07 INFO rasa_sdk.endpoint - Action endpoint is up and running on http://0.0.0.0:5055
```

Fig. 8.2: Rasa action server terminal screen

- **Rasa server screen**

The Rasa Server Screen refers to the console or command line interface that displays information and logs related to the Rasa server. The Rasa server is responsible for handling user interactions, managing conversations, and coordinating the dialogue flow in your Rasa chatbot.

Here's an explanation of the key components you may find on the Rasa Server Screen:

**Startup Information:** When you start the Rasa server, the terminal screen typically displays information about the server's startup process. This includes details such as the server address, port number, and any other configurations you have specified.

**User Input and Bot Responses:** As the Rasa server runs, it logs information about the user inputs received and the corresponding responses generated by the chatbot. It shows the messages exchanged between the user and the chatbot, along with additional details like intents, entities, and any other relevant information associated with each message.

**Intent Recognition and Entity Extraction:** The server screen may display logs related to the intent recognition and entity extraction performed by the Rasa NLU (Natural Language Understanding) component. It shows the recognized intents, extracted entities, and confidence scores for each user message processed.

**Dialogue Management and Actions:** The server screen provides logs related to the dialogue management and actions taken by the Rasa Core component. It shows the predicted actions, policies used, and the conversation history that Rasa Core uses to make predictions and generate responses.

**Training and Model Loading:** If you have trained your Rasa models and loaded them into the server, the screen may display logs related to model loading, including the model name, version, and any relevant training information.

**Error Messages and Debugging:** If there are any errors or exceptions encountered during the server's operation, the terminal screen displays error messages and traceback information. These messages help you identify and debug issues in your chatbot's configuration, training data, or model setup.

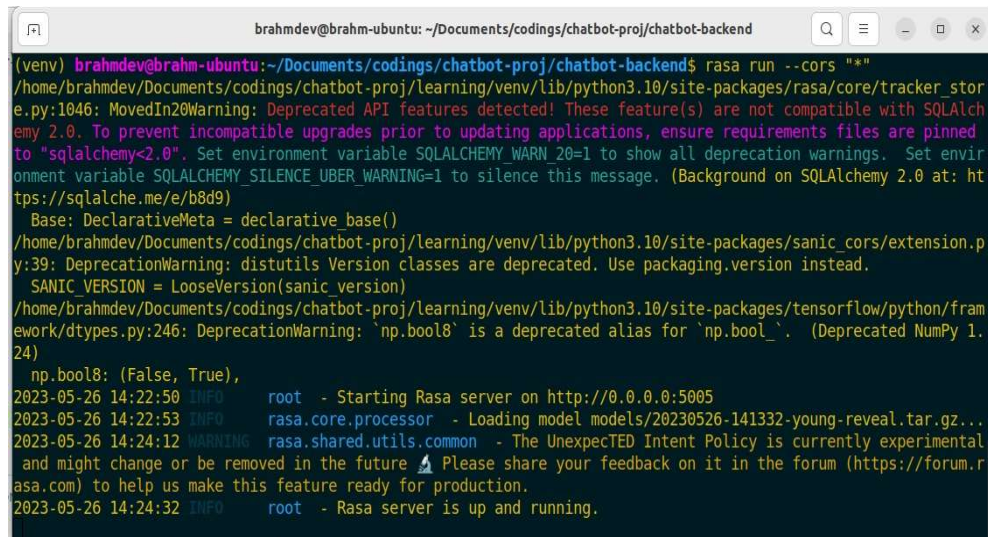
**Server Status and Health Checks:** The terminal screen may include status information about the Rasa server, such as its health or connectivity to external services. It can display logs related to any health checks being performed or connection attempts made by the server.

By monitoring the Rasa Server Screen, you can observe the user interactions, track the conversation flow, debug any errors, and gain insights into the behaviour of your Rasa chatbot.

It is important to note that the specific format and details of the server screen may vary depending on your operating system, terminal configuration, and the version of Rasa you are using. The logs displayed on the server screen provide valuable information for troubleshooting, understanding user interactions, and evaluating the performance of your Rasa chatbot.



The trained model saved inside the *models* directory is loaded and run using the command `rasa run --cors "*"` . The server is started and runs on the default port number 5005 with the api link <http://0.0.0.0:5005>. The queries to the bot are sent on this address in the form of post request.

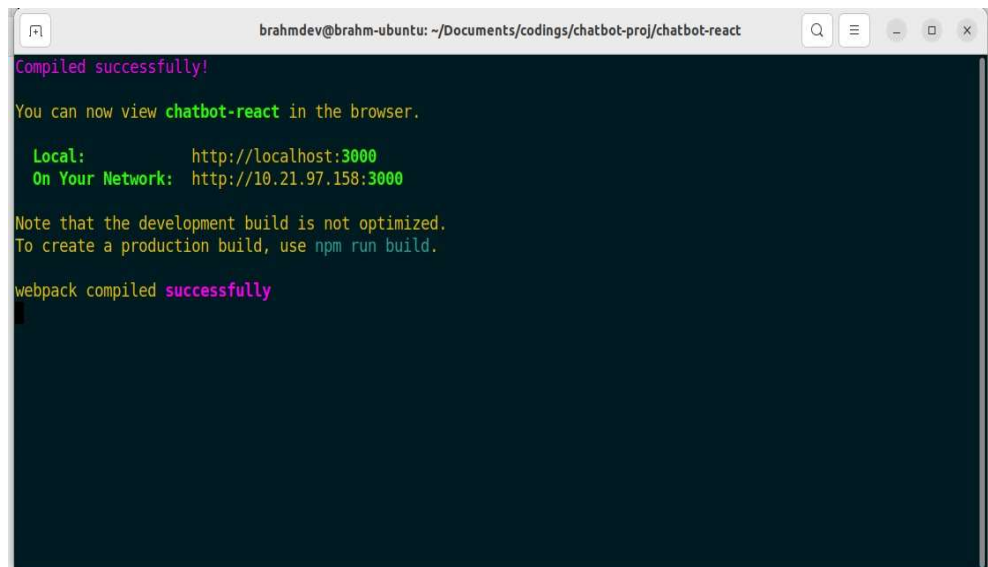


```
brahmdev@brahm-ubuntu: ~/Documents/codings/chatbot-proj/chatbot-backend
(venv) brahmdev@brahm-ubuntu:~/Documents/codings/chatbot-proj/chatbot-backend$ rasa run --cors "*"
/home/brahmdev/Documents/codings/chatbot-proj/learning/venv/lib/python3.10/site-packages/rasa/core/tracker_store.py:1046: MovedIn20Warning: Deprecated API features detected! These feature(s) are not compatible with SQLAlchemy 2.0. To prevent incompatible upgrades prior to updating applications, ensure requirements files are pinned to "sqlalchemy<2.0". Set environment variable SQLALCHEMY_WARN_20=1 to show all deprecation warnings. Set environment variable SQLALCHEMY_SILENCE_UBER_WARNING=1 to silence this message. (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  Base: DeclarativeMeta = declarative_base()
/home/brahmdev/Documents/codings/chatbot-proj/learning/venv/lib/python3.10/site-packages/sanic_cors/extension.py:39: DeprecationWarning: distutils Version classes are deprecated. Use packaging.version instead.
  SANIC_VERSION = LooseVersion(sanic version)
/home/brahmdev/Documents/codings/chatbot-proj/learning/venv/lib/python3.10/site-packages/tensorflow/python/framework/dtypes.py:246: DeprecationWarning: 'np.bool8' is a deprecated alias for 'np.bool_'. (Deprecated NumPy 1.24)
  np.bool8: (False, True),
2023-05-26 14:22:50 INFO root - Starting Rasa server on http://0.0.0.0:5005
2023-05-26 14:22:53 INFO rasa.core.processor - Loading model models/20230526-141332-young-reveal.tar.gz...
2023-05-26 14:24:12 WARNING rasa.shared.utils.common - The Unexpected Intent Policy is currently experimental and might change or be removed in the future. Please share your feedback on it in the forum (https://forum.rasa.com) to help us make this feature ready for production.
2023-05-26 14:24:32 INFO root - Rasa server is up and running.
```

Fig. 8.3: Rasa server screen

- **Starting the frontend**

The frontend in ReactJS is started using the command `npm start`. The frontend of the bot is started and runs on the default port 3000. The frontend is accessible on the address <http://localhost:3000>.



```
brahmdev@brahm-ubuntu: ~/Documents/codings/chatbot-proj/chatbot-react
Compiled successfully!

You can now view chatbot-react in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://10.21.97.158:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

Fig. 8.4: Starting the frontend



- **Chatbot interaction screen**

The frontend screen of the bot is accessible on the address <http://localhost:3000>.  
The screenshot of a simple interaction with the bot is attached below.



Fig. 8.5: Chatbot interaction screen1

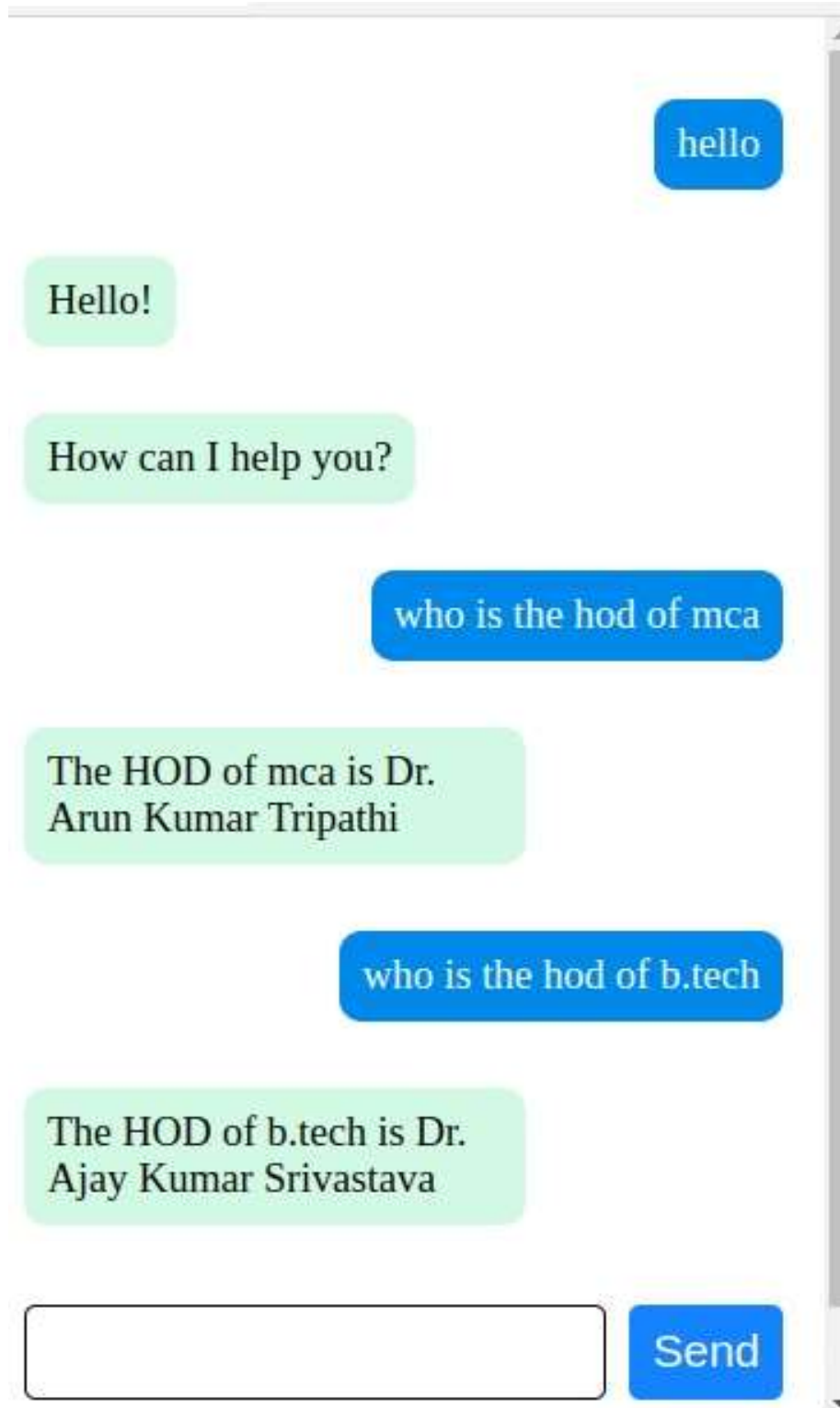


Fig. 8.6: Chatbot interaction screen2



Fig. 8.7: Chatbot interaction screen3

## 8.5 Code

### config.yml

```
recipe: default.v1
assistant_id: 20230402-231449-denim-audit
language: en
pipeline: null
# # No configuration for the NLU pipeline was provided. The following default
# pipeline was used to train your model.
# # If you'd like to customize it, uncomment and adjust the pipeline.
# # See https://rasa.com/docs/rasa/tuning-your-model for more information.
# - name: WhitespaceTokenizer
# - name: RegexFeaturizer
# - name: LexicalSyntacticFeaturizer
# - name: CountVectorsFeaturizer
# - name: CountVectorsFeaturizer
#   analyser: char_wb
#   min_ngram: 1
#   max_ngram: 4
# - name: DIETClassifier
#   epochs: 100
#   constrain_similarities: true
# - name: EntitySynonymMapper
# - name: ResponseSelector
#   epochs: 100
#   constrain_similarities: true
# - name: FallbackClassifier
#   threshold: 0.3
#   ambiguity_threshold: 0.1
policies: null
# # No configuration for policies was provided. The following default policies
# were used to train your model.
# # If you'd like to customize them, uncomment and adjust the policies.
# # See https://rasa.com/docs/rasa/policies for more information.
# - name: MemoizationPolicy
# - name: RulePolicy
# - name: UnexpectEDIntentPolicy
#   max_history: 5
#   epochs: 100
# - name: TEDPolicy
#   max_history: 5
#   epochs: 100
#   constrain_similarities: true
```

## credentials.yml

```
# This file contains the credentials for the voice & chat platforms
# which your bot is using.
# https://rasa.com/docs/rasa/messaging-and-voice-channels

rest:
# # you don't need to provide anything here - this channel doesn't
# # require any credentials

#facebook:
# verify: "<verify>"
# secret: "<your secret>"
# page-access-token: "<your page access token>"

#slack:
# slack_token: "<your slack token>"
# slack_channel: "<the slack channel>"
# slack_signing_secret: "<your slack signing secret>"

# socketio:
# user_message_evt: user_uttered
# bot_message_evt: bot_uttered
# session_persistence: true

#mattermost:
# url: "https://<mattermost instance>/api/v4"
# token: "<bot token>"
# webhook_url: "<callback URL>"

# This entry is needed if you are using Rasa Enterprise. The entry represents
# credentials
# for the Rasa Enterprise "channel", i.e. Talk to your bot and Share with guest
# testers.
rasa:
url: http://localhost:5002/api
```

## Dockerfile

```
FROM python:3.10.6 AS BASE

WORKDIR /app

ADD . /app/

RUN pip install --no-cache-dir --upgrade pip

RUN pip install -r requirements.txt
```

## endpoints.yml

```
# This file contains the different endpoints your bot can use.

# Server where the models are pulled from.
# https://rasa.com/docs/rasa/model-storage#fetching-models-from-a-server

#models:
# url: http://my-server.com/models/default_core@latest
# wait_time_between_pulls: 10 # [optional](default: 100)

# Server which runs your custom actions.
# https://rasa.com/docs/rasa/custom-actions

action_endpoint:
  url: "http://localhost:5055/webhook"
# Tracker store which is used to store the conversations.
# By default the conversations are stored in memory.
# https://rasa.com/docs/rasa/tracker-stores

#tracker_store:
# type: redis
# url: <host of the redis instance, e.g. localhost>
# port: <port of your redis instance, usually 6379>
# db: <number of your database within redis, e.g. 0>
# password: <password used for authentication>
# use_ssl: <whether or not the communication is encrypted, default false>

#tracker_store:
# type: mongod
# url: <url to your mongo instance, e.g. mongodb://localhost:27017>
```

```
# db: <name of the db within your mongo instance, e.g. rasa>
# username: <username used for authentication>
# password: <password used for authentication>

# Event broker which all conversation events should be streamed to.
# https://rasa.com/docs/rasa/event-brokers

#event_broker:
# url: localhost
# username: username
# password: password
# queue: queue
```

## Okteto.yml

```
version: "1.0"

services:
  action-server:
    image: rasa-bot:latest
    working_dir: /app
    build: "./"
    restart: always
    volumes:
      - ./actions:/app/actions
      - ./data:/app/data
    command: ["rasa", "run", "actions"]
    ports:
      - "5055:5055"
    public: true
    networks:
      - all

  rasa-server:
    image: rasa-bot:latest
    working_dir: /app
    build: "./"
    restart: always
    volumes:
      - ./actions:/app/actions
      - ./data:/app/data
    command: bash -c "rm -rf /app/models/* && rasa train && rasa run --enable-api --cors \"*\" --debug -p 5005"
```

```
ports:
  - "5005:5005"
public: true
networks:
  - all

networks:
  all:
    driver: bridge
    driver_opts:
      com.docker.network.enable_ipv6: "true"
```

### **requirements.txt**

```
rasa==3.5.3
```

### **domain.yml**

```
version: "3.1"

intents:
  - greet
  - goodbye
  - fee_structure
  - fee_misc
  - admission_process
  - hod
  - placement
  - random
  - ask_name

slots:
  course:
    type: text
    influence_conversation: true
    mappings:
      - type: from_entity
        entity: course

department:
  type: text
  influence_conversation: true
```



```

mappings:
  - type: from_entity
    entity: department

entities:
  - course
  - department

actions:
  - action_utter_fees
  - action_utter_hod

responses:
  utter_greet:
    - text: "Hello!"
    - text: "Hi, there!"
    - text: "Hi!"

  utter_goodbye:
    - text: "Bye"
    - text: "Bye! Take care!"
    - text: "See you soon!"

  utter_help:
    - text: "How can I help you?"

  utter_fees_misc:
    - text: "Sorry, thesedetails have not been added yet."

  utter_admission_process:
    - text: "Sorry, admission process details have not been added yet."

  utter_placement:
    - text: "KIET Group of Institutions has the best placements in Delhi NCR."

  utter_random:
    - text: "I have not been trained to answer that."

  utter_name:
    - text: "I am Bot."

session_config:
  session_expiration_time: 60
  carry_over_slots_to_new_session: true

```

## **data/nlu.yml**

version: "3.1"

nlu:

- intent: greet

examples: |

- hey
- hello
- hi
- hello there
- good morning
- good evening
- moin
- hey there
- let's go
- hey dude
- goodmorning
- goodevening
- good afternoon

- intent: ask\_name

examples: |

- what is your name
- who are you
- who is you
- tell your name
- your name

- intent: goodbye

examples: |

- cu
- good by
- cee you later
- good night
- bye
- goodbye
- have a nice day
- see you around
- bye bye
- see you later

- intent: fee\_structure

examples: |

- what is the fees of [mca](course)
  - what is [b. tech](course) fees?
  - [mca](course) fee?
  - [b. pharma](course) admission fees
  - [mba](course) yearly expenditure
  - [hostel](course) fees
  - [mca](course) semester wise fees
  - What is the fee structure of the [hostel](course)?
  - fees for [mca](course)
  - What are the [hidden](course) fees?
  - what is the fees in [mca](course)
  - what is the fees in [b.tech](course) [cse](branch)
- intent: fee\_misc  
examples: |
- Is there any tax in doing online fees submission?
  - Can I deposit fees in Cash?
  - Can I deposit fees using Netbanking?
  - How much is the late fine?
  - How much is the Library charge?
  - How much is the security amount?
  - Is the course fees refundable?
  - Is there any fee concessions for students who belong to weaker economical section?
  - Is there any Installment scheme?
  - Are there any scholarships or fee waivers in KIET Group of Institutions?
  - Last date of fees submission?
  - What are hidden fees?
  - Is there any tax in doing online fees submission?
  - Is there any fee concessions for students who belong to weaker economical section?
  - Is there any Installment scheme?
  - what is fees of mca
  - Can I deposit Hostel fees using Netbanking?
  - How much is the late fine?
  - Last date of fees submissions?
  - Can I leave hostel in between semesters?
  - How much is the security amount?
  - Is the hostel fees refundable?
  - Can I deposit Hostel fees in Cash?
  - What is the fee structure of MCA.
- intent: admission\_process  
examples: |

- documents required for mca
- whom to contact for mca admission
- whom to contact for b.tech enrollment
- is kiet is taking online interview for admission in btech , can anybody tell me what questions they could ask?
- Fee is same or different for all Courses?
- Value added courses runs in KIET?
- KIET is aided or fully private Institution?
- KIET Group of Institution is founded in which year?
- KIET is Affiliated by Which university?
- Alumini Feedback?
- Sponsr's and partner of KIET?
- KIET Group of Institution run's by?
- Ranking of KIET Group of Institution among all colleges in Delhi NCR?
- No . of Labs available in KIET in diffrent Departments?
- No . of Seats available in Different courses in KIET?
- Whom to contact for process?
- admission process of b.pharma
- how to get admission in b.tech?
- how to get scholarship
- what is the eligibility criteria for scholarship
- how to get admission in kiet group of institutions
- Semester length of courses in kiet ?
- Result summary of courses?
- how many mca
- Hackathons organised by KIET?
- Courses Offered by KIET Ghaziabad ?
- Competition organised by KIET?
- KIET Group of Institutions Admission Process?
- KIET Ghaziabad Cutoff For distinct courses?
- KIET Ghaziabad Campus and Facilities?
- KIET contact No for admission?
- Is thier any management quota seats is available in kiet ghaziabad?
- What is the Admission cancellation procudeure?
- Attendance mandate of courses in kiet college?
- How can I take direct admission in Kiet Ghaziabad?
- Documents required during the counselling?
- College working days?
- Departmental activies and functions?
- Academics Hours in KIET?
- Clubs and societies in KIET?
- is kiet Offeres any international trip or tour for students?
- Documents required for getting direct Admission in KIET ?
- is kiet celebrates any Annnual function or Event?

- Naac Grading of KIET?
  
- intent: hod
  - examples: |
    - who is the hod of [mba](department)
    - who is the hod of [mca](department)
    - who is the hod of [b.pharm](department)
    - who is the hod of [b.tech](department)
  
- intent: placement
  - examples: |
    - Are there placement coordinators appointed in department?
    - What is the minimum package in CSE branch?
    - Can I get good package from CSIT from KIET Group of institutions?
    - Is KIET Ghaziabad good for EEE?
    - How are the placements for mechanical branch at KIET Ghaziabad?
    - Which companies come for placements at KIET for B.Tech?
    - Who were the recruiters at KIET Ghaziabad placements 2022?
    - What was the average salary of KIET Ghaziabad?
    - What was the average placement of KIET?
    - What is duration of internships offered?
    - How are the placements at KIET, Ghaziabad?
    - What is better, IT from KIET Ghaziabad or CSE from ABES Engineering college Ghaziabad?
    - How are the placements in KIET Ghaziabad for a Bachelor's of Technology students?
    - Is the ECE branch placement at ABES Engineering College better than at KIET?
    - Are there any internships offered to Civil students?
    - What is the highest package received?
    - whom to contact for admission in b.pharma
    - What is the use of CRPC?
    - Do I get paid internships after B. Pharma from kiet Ghaziabad?
    - Are unpaid internships available after MBA from KIET Ghaziabad?
    - What are the mass recruiters in each department of KIET?
    - What other additional facilities are provided for placements in KIET MBA?
    - Do more companies visit KIET Ghaziabad or AKG Ghaziabad?
    - Which companies were major recruiters at KIET, Ghaziabad?
    - What is the Placement Statics at KIET Ghaziabad?
    - Does KIET Ghaziabad provide Internship Programs?
    - Is KIET Ghaziabad a good Institute/College?
    - Are the placements of KIET Ghaziabad getting better year by year?
    - What are the reasons for KIET Ghaziabad's low placement rate?
    - Best company for placement in KIET?

- How are KIET Ghaziabad placements 2018?
- Which department is best for high package job MCA or MBA?
- Which branch of B. Tech. gets the highest placements in KIET?
- What is the highest package in KIET till now?
- Can I get placed in Amazon after MCA from KIET Ghaziabad?
- Is any additional payment related for required for placement after B. Tech?
- Is KIET Ghaziabad good for Higher studies?
- Do I get Aptitude and verbal ability classes in MCA?
- Are there additional skill development classes in B. Pharma?
- Which companies come for recruitment in MBA in KIET Ghaziabad?
- Is MCA placement better at KIET than AKGEC, GL Bajaj and ABES?
- Are there any unplaced students after completion of MCA from KIET Ghaziabad?

- intent: random
- examples: |
  - who is Anuj
  - do you eat
  - do you have feelings
  - how to go to goa
  - who is the pm of china

#### **data/rules.yml**

version: "3.1"

rules:

- rule: Say goodbye anytime the user says goodbye
- steps:
  - intent: goodbye
  - action: utter\_goodbye

#### **data/stories.yml**

version: "3.1"

stories:

- story: happy path
- steps:
  - intent: greet
  - action: utter\_greet
  - action: utter\_help

- story: ask name  
steps:
  - intent: ask\_name
  - action: utter\_name
  
- story: fees enquiry  
steps:
  - intent: fee\_structure
  - action: action\_utter\_fees
  
- story: miscellaneous fees enquiry details  
steps:
  - intent: fee\_misc
  - action: utter\_fees\_misc
  
- story: fees enquiry with greet  
steps:
  - intent: greet
  - action: utter\_greet
  - intent: fee\_structure
  - action: action\_utter\_fees
  
- story: admission enquiry  
steps:
  - intent: admission\_process
  - action: utter\_admission\_process
  
- story: hod enquiry  
steps:
  - intent: hod
  - action: action\_utter\_hod
  
- story: placement enquiry  
steps:
  - intent: placement
  - action: utter\_placement
  
- story: random questions  
steps:
  - intent: random
  - action: utter\_random

## actions/actions.py

```
# This file contains your custom actions which can be used to run
# custom Python code.
#
# See this guide on how to implement these actions:
# https://rasa.com/docs/rasa/custom-actions

# This is a simple example for a custom action which utters "Hello World!"

from typing import Any, Text, Dict, List

from rasa_sdk import Action, Tracker
from rasa_sdk.executor import CollectingDispatcher

#
#
# class ActionHelloWorld(Action):
#
#     def name(self) -> Text:
#         return "action_hello_world"
#
#     def run(self, dispatcher: CollectingDispatcher,
#             tracker: Tracker,
#             domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
#
#         dispatcher.utter_message(text="Hello World!")
#
#         return []

fees = {
    'mca': 138499,
    'b.tech': 151239,
    'mba': 122890
}

hod = {
    'mca': "Dr. Arun Kumar Tripathi",
    'b.tech': "Dr. Ajay Kumar Srivastava",
    'mba': "Dr. Atif Ali",
    'b.pharma': "Dr. Anuj"
}

class ActionUtterFees(Action):
```



```

def name(self) -> Text:
    return "action_utter_fees"

def run(self, dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

    course = tracker.get_slot('course')
    dispatcher.utter_message(text=f"The fees of {course} is {fees[course]}")
    return []

```

```

class ActionUtterFees(Action):

```

```

    def name(self) -> Text:
        return "action_utter_hod"

    def run(self, dispatcher: CollectingDispatcher,
            tracker: Tracker,
            domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        dept = tracker.get_slot('department')
        dispatcher.utter_message(text=f"The HOD of {dept} is {hod[dept]}")
        return []

```

## Frontend

### package.json

```

{
  "name": "chatbot-react",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^1.2.5",
    "form-data": "^4.0.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",

```

```

    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  },
  "homepage": "."
}

```

### **public/index.html**

```

<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Chat App</title>
</head>

<body>
  <div id="root">

```

```
</div>
</body>

</html>
```

### **src/index.js**

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";
import "./index.css";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

### **src/index.css**

```
/* for desktop screens */
body {
  position: absolute;
  right: 0px;
  bottom: 0px;
  font-size: 18px;
  border-radius: 10px;
  height: 600px;
  width: 350px;
}

.chats-div {
  margin: 10px;
}

.query-div {
  margin-bottom: 30px;
  display: flex;
  justify-content: flex-end;
}
```

```

.query-bubble {
  width: fit-content;
  max-width: 60%;
  padding: 10px;
  border-radius: 10px;
  background-color: rgb(0, 137, 235);
  color: white;
}

.reply-div {
  margin-bottom: 30px;
  display: flex;
  justify-content: flex-start;
}

.reply-bubble {
  width: fit-content;
  max-width: 60%;
  padding: 10px;
  border-radius: 10px;
  background-color: rgb(210, 249, 227);
  color: black;
}

#input-div {
  position: fixed;
  display: flex;
  bottom: 0px;
  padding: 10px 10px 10px 10px;
  justify-content: space-between;
  width: 330px;
  background-color: white;
}

#msg-input {
  padding: 10px;
  color: black;
  font-size: 18px;
  width: 70%;
  border: 1px solid black;
  border-radius: 5px;
}

```

```

#msg-input:focus {
  outline-color: blue;
}

#send-button {
  padding: 10px;
  height: min-content;
  font-size: 20px;
  background-color: rgb(21, 130, 255);
  color: white;
  border: none;
  border-radius: 5px;
}

/* for mobile screens */
@media only screen and (max-width: 600px) {
  body {
    border: none;
    font-size: 20px;
    width: 100%;
    height: 100%;
  }

  .chats-div {
    height: 70%;
    margin: 10px;
  }

  .query-div {
    margin-bottom: 30px;
    display: flex;
    justify-content: flex-end;
  }

  .query-bubble {
    width: fit-content;
    max-width: 60%;
    padding: 10px;
    border-radius: 10px;
    background-color: rgb(0, 137, 235);
    color: white;
  }
}

```

```

.reply-div {
  margin-bottom: 30px;
  display: flex;
  justify-content: flex-start;
}

.reply-bubble {
  width: fit-content;
  max-width: 60%;
  padding: 10px;
  border-radius: 10px;
  background-color: rgb(210, 249, 227);
  color: black;
}

#input-div {
  position: fixed;
  display: flex;
  max-width: 100%;
  bottom: 0px;
  padding-left: 20px;
  justify-content: space-between;
  background-color: white;
}

#msg-input {
  padding: 10px;
  color: black;
  font-size: 20px;
  width: 70%;
  border: 1px solid black;
  border-radius: 5px;
}

#msg-input:focus {
  outline-color: blue;
}

#send-button {
  padding: 10px;
  height: min-content;
  font-size: 20px;
  background-color: rgb(21, 130, 255);
  color: white;
}

```

```

border: none;
border-radius: 5px;
}
}

```

### **src/ChatBubble.jsx**

```

function ChatBubble(props) {
  return (
    <div>
      {props.msg.type === "query" ? (
        <QueryBubble msg={props.msg} />
      ) : (
        <ReplyBubble msg={props.msg} />
      )}
    </div>
  );
}

function QueryBubble(props) {
  return (
    <div className="query-div">
      <div className="query-bubble">{props.msg.text}</div>
    </div>
  );
}

function ReplyBubble(props) {
  return (
    <div className="reply-div">
      <div className="reply-bubble">{props.msg.text}</div>
    </div>
  );
}

export default ChatBubble;

```

### **src/app.js**

```

import { useState } from "react";
import ChatBubble from "../ChatBubble";
import axios from "axios";

```

```

import { useRef, useEffect } from "react";

function App() {
  const [text, setText] = useState("");
  const BOT_URL = "http://localhost:5005/webhooks/rest/webhook";
  const messageRef = useRef();

  let initialChats = [];

  const [chats, setChats] = useState(initialChats);

  useEffect(() => {
    if (messageRef.current) {
      messageRef.current.scrollToView({
        behavior: "smooth",
        block: "end",
        inline: "nearest",
      });
    }
  }, [chats]);

  const updateText = (event) => {
    setText(event.target.value);
  };

  const sendMessage = () => {
    if (text === "") return;
    const newChats = [
      ...chats,
      {
        id: 11,
        type: "query",
        time: new Date(),
        text: text,
      },
    ];
    setChats(newChats);
    getResponse(text, newChats);
    setText("");
  };

  const getResponse = async (msg, newChats) => {
    var bodyFormData = { message: msg };
    const response = axios.post(BOT_URL, bodyFormData);
    var messages = [];
  }

```



```

response
  .then((res) => {
    console.log(res);
    messages = [];
    res.data.map(
      (data) =>
        (messages = [
          ...messages,
          {
            id: 12,
            type: "reply",
            time: new Date(),
            text: data.text,
          },
        ])
    );
    setChats([...newChats, ...messages]);
  })
  .catch((error) => console.log("Error in response: " + error));
};
return (
  <div className="container">
    <div className="chats-div" id="chats-div" ref={messageRef}>
      {chats.map((chat) => (
        <ChatBubble key={chat.id} msg={chat} />
      ))}
    <div style={{ height: "100px" }} id="divider"></div>
  </div>
  <div id="input-div">
    <input
      type="text"
      name="msg"
      value={text}
      id="msg-input"
      onInput={updateText}
    />
    <button id="send-button" onClick={sendMessage}>
      Send
    </button>
  </div>
</div>
);
}
export default App;

```

## CHAPTER 9: MODULES DESCRIPTION

### Frontend

- **Chat** – The screen with input UI for interacting with the bot by sending queries and getting message replies like a traditional messenger application.
- **See previous chats** – For viewing the previous chats with the bot stored on the cloud. This reduces additional overhead on the bot server as the user does not have to ask the same query repeatedly.

### Backend

- **Chat intent classification** - for classifying the category of the question being asked to map it to the correct category and choose the most appropriate answer to be given.
- **Machine Learning model for answering queries** - the Rasa NLU model combined with Rasa core to perform Entity Extraction and compose the complete answer using the custom action server.
- **Rasa Action Server** - Action server in Python to get answers of questions that require some condition making or getting data from any API.

## CHAPTER 10: TESTING

### Chatbot testing criteria

Criteria derived from the Rasa framework, which is a set of commands that are executed to obtain degrading results based on the contents and how the bot is built. It is done by executing specific commands, data validation verifies that no major errors or inconsistencies appear in your domain, NLU data, or story data. If data validation leads to errors, training the model may also fail or result in poor performance, so it is always a good idea to do this check before training the model. Including the *--fail-on-warnings* flag, this step will fail warnings that indicate more minor issues.

Another thing you can do to get the bot to be tested is to write test stories. Testing your trained model on test stories is the best way to be confident in how your assistant will behave in certain situations. Test stories, written in a modified story format, allow you to present complete conversations and test that, given certain user input, your model will behave expectedly. This is especially important when you start to present more complex stories than user conversations. Test stories are similar to the stories in your training data but include the user's message as well.

**Conversation flow:** In this criterion, the following things should be checked while testing:

- Does a chatbot clearly understand the questions?
- Does it always give instant responses to these questions?
- Are the answers relevant to the given questions?
- Should a user ask a series of questions until he gets the answer?
- Does a chatbot engage the user to continue the conversation?

**Business-specific questions:** Each industry has its specific terminology, notions, nuances. So, a tester should have a list of domain-specific questions to check if the chatbot is able to answer those.

**Confusion handling:** Confusion may arise if a user enters some expression with double meaning or an unknown word for a chatbot. The latter should be taught to answer in such a situation. The tester's goal is to check if the chatbot can handle misunderstandings, exceptional conversational scenarios, and unusual patterns. This capability of a chatbot is showing how “emotionally intelligent” it is.

The testing process involved the following steps:

- Understand the Requirements: We thoroughly reviewed the functional and non-functional requirements of the chatbot to gain a clear understanding of its intended functionality and behaviour.
- Identify Test Scenarios: Based on the requirements, we identified various test scenarios covering different aspects of the chatbot's functionality.

- Design Test Cases: Detailed test cases were designed for each identified test scenario, outlining the steps to be executed, expected results, and actual results.
- Prioritize Test Cases: Test cases were prioritized based on criticality, complexity, and frequency of use to ensure efficient testing.
- Implement Test Automation: Test automation was implemented using Rasa Test Core and React Testing Library to automate repetitive tasks and enhance test coverage.
- Execute Test Cases: The designed test cases were executed, and the actual results were recorded and compared against the expected results.
- Report and Track Issues: Any bugs or issues discovered during testing were documented with detailed steps to reproduce and supporting information.
- Retest and Regression Testing: Retesting was performed to validate the fixes for identified issues, and regression testing was conducted to ensure existing functionality was not impacted.
- Continuous Testing: Continuous testing practices were incorporated to ensure ongoing quality assurance as the chatbot evolves.

## **Test cases**

Test Case 1: User Query about HOD of a department

- Test Case Description: Verify if the chatbot provides accurate information about department HOD.
- Test Steps:
  1. User enters the query, "Who is the HOD of MCA?"
  2. Chatbot processes the query and provides a response.
- Expected Results: "Dr. Arun Kumar Tripathi"
- Actual Results: "The HOD of the MCA is Dr. Arun Kumar Tripathi."
- Pass/Fail: Pass

Test Case 2: User Query about a course fee

- Test Case Description: Verify if the chatbot provides accurate information about yearly fees of the asked course
- Test Steps:
  1. User enters the query, "What is the fees of MCA?"
  2. Chatbot processes the query and provides a response.
- Expected Results: "138999"
- Actual Results: "The fees of MCA is 138999"
- Pass/Fail: Pass

Test Case 3: User Query about library fees

- Test Case Description: Verify if the chatbot provides accurate information about fees of the library
- Test Steps:
  1. User enters the query, "What are the charges of library?"
  2. Chatbot processes the query and provides a response.

- Expected Results: “No additional charges”
- Actual Results: “There is no additional charge for library, it is included in the fees.”
- Pass/Fail: Pass

#### Test Case 4: User Query about placement status

- Test Case Description: Verify if the chatbot provides accurate information about placement status
- Test Steps:
  1. User enters the query, "How are placement at KIET Group of Institutions?"
  2. Chatbot processes the query and provides a response.
- Expected Results: The chatbot should give statistics of placement in the college
- Actual Results: “KIET Group of Institutions provides the best placement in Delhi NCR.”
- Pass/Fail: Fail (because it does not provide current placement statistics)

#### Test Case 5: User Query about Hostel fees

- Test Case Description: Verify if the chatbot provides accurate information about yearly hostel fees
- Test Steps:
  1. User enters the query, "What is the hostel fees for complete one year?"
  2. Chatbot processes the query and provides a response.
- Expected Results: “96000”
- Actual Results: “The charges of hostel per year is 96000.”
- Pass/Fail: Pass

#### Test Case 6: User Query about AC/Non-AC rooms in hostel

- Test Case Description: Verify if the chatbot provides accurate information about yearly fees of the asked course
- Test Steps:
  1. User enters the query, "Is there facility of AC rooms in hostel?"
  2. Chatbot processes the query and provides a response.
- Expected Results: “Yes”
- Actual Results: “Yes, there are both types of room available in the hostel.”
- Pass/Fail: Pass

#### Test Case 7: User greets the bot

- Test Case Description: Verify if the chatbot replies to greetings
- Test Steps:
  1. User enters the query, "Hello!"
  2. Chatbot processes the query and provides a response.
- Expected Results: The chatbot replies with some greetings
- Actual Results: “Hi! How can I help you?”
- Pass/Fail: Pass

Test Case 8: User Query with some random question

- Test Case Description: Verify if the chatbot answer random questions

- Test Steps:

1. User enters the query, "How can I get full marks in CTs?"

2. Chatbot processes the query and provides a response.

- Expected Results: The chatbot should deny the response

- Actual Results: "Sorry, I have not been trained to answer these questions."

- Pass/Fail: Pass

Test Case 9: User Query about admission process

- Test Case Description: Verify if the chatbot provides accurate information about how to get admissions in the college

- Test Steps:

1. User enters the query, "How can I get admission in KIET Group of Institutions?"

2. Chatbot processes the query and provides a response.

- Expected Results: The chatbot should provide information about both direct and through CUET exam modes

- Actual Results: "There are two ways to get admission: 1. Direct admission 2. Through CUET examination"

- Pass/Fail: Pass

Test Case 10: User Query about fee payment process

- Test Case Description: Verify if the chatbot provides accurate information about fee payment process

- Test Steps:

1. User enters the query, "How do I pay the fees?"

2. Chatbot processes the query and provides a response.

- Expected Results: The chatbot should provide the process to pay the fees while admission

- Actual Results: "Fees can be paid in both online and offline modes"

- Pass/Fail: Fail (because it does not provide the details of the fee payment process)

## CHAPTER 11: SCOPE FOR FUTURE IMPROVEMENTS

In this project built using ReactJS and Rasa, there are several future improvement scopes that can enhance the capabilities and user experience. Here are some potential areas to focus on:

### **1. Natural Language Understanding (NLU) Enhancement:**

Improving the chatbot's NLU capabilities is crucial for better understanding user inputs. This can involve expanding the training data, refining entity extraction, and optimizing intent classification models. Additionally, exploring advanced techniques such as transfer learning or incorporating pre-trained language models like BERT or GPT-3 can enhance the chatbot's ability to comprehend user queries accurately.

### **2. Contextual Understanding and Memory:**

Enabling the chatbot to maintain context across conversations can significantly improve user interactions. Future improvements can include implementing a memory component that allows the chatbot to remember user preferences, previous queries, or conversation history. This context can be utilized to provide more personalized and relevant responses, leading to a more natural and engaging conversation flow.

### **3. Multilingual Support:**

Extending the chatbot's language capabilities can enhance its accessibility and usefulness. Integrating language translation services or training models to understand and respond in multiple languages can broaden the chatbot's reach and cater to a more diverse user base.

### **5. Rich Media Support:**

Enhancing the chatbot's ability to handle rich media content, such as images, videos, or documents, can make interactions more engaging and informative. Implementing features like media input parsing, image recognition, or multimedia response generation can enable the chatbot to process and provide relevant information using different types of media.

### **6. Continuous Learning and Training:**

Enabling the chatbot to learn and improve over time can be valuable. Implementing mechanisms for continuous learning, such as user feedback collection, active learning, or reinforcement learning, can help the chatbot adapt to user preferences and refine its responses based on user interactions. This iterative learning process can lead to a more intelligent and effective chatbot.

### **7. User Interface (UI) and User Experience (UX) Enhancements:**

Focusing on improving the UI and UX of the chatbot can make interactions more intuitive and user-friendly. Future improvements can involve refining the design,

implementing conversational UI components, adding visual cues, or providing progress indicators during longer processes. Attention to UI/UX can enhance user satisfaction and encourage more seamless interactions with the chatbot.

By considering these future improvement scopes, a chatbot project built using React and Rasa can evolve into a more intelligent, context-aware, and user-centric conversational agent. These enhancements can contribute to a more engaging and effective user experience across various domains and use cases.



## REFERENCES

### E-books

1. “Natural Language Processing in Artificial Intelligence” by Brojo Kishore Mishra  
[https://books.google.co.in/books?hl=en&lr=&id=tAn-DwAAQBAJ&oi=fnd&pg=PP1&dq=Natural+Language+Processing+in+Artificial+Intelligence+Edited+ByBrojo+Kishore+Mishra,+Raghvendra+Kumar&ots=jZgTTIWJT6&sig=ifbcy0xxRI2Gx8V0EdVLja2boM&redir\\_esc=y#v=onepage&q=Natural%20Language%20Processing%20in%20Artificial%20Intelligence%20Edited%20ByBrojo%20Kishore%20Mishra%20C%20Raghvendra%20Kumar&f=false](https://books.google.co.in/books?hl=en&lr=&id=tAn-DwAAQBAJ&oi=fnd&pg=PP1&dq=Natural+Language+Processing+in+Artificial+Intelligence+Edited+ByBrojo+Kishore+Mishra,+Raghvendra+Kumar&ots=jZgTTIWJT6&sig=ifbcy0xxRI2Gx8V0EdVLja2boM&redir_esc=y#v=onepage&q=Natural%20Language%20Processing%20in%20Artificial%20Intelligence%20Edited%20ByBrojo%20Kishore%20Mishra%20C%20Raghvendra%20Kumar&f=false)
2. “Graph-based Natural Language Processing and Information Retrieval”  
[https://books.google.co.in/books?hl=en&lr=&id=kByP4X9c5AQC&oi=fnd&pg=PA1&dq=Graph-based+Natural+Language+Processing+and+Information+Retrieval&ots=\\_EleaeagYQ&sig=zsZwm5PCHToDjARJbtyVhze5q-E&redir\\_esc=y#v=onepage&q=Graph-based%20Natural%20Language%20Processing%20and%20Information%20Retrieval&f=true](https://books.google.co.in/books?hl=en&lr=&id=kByP4X9c5AQC&oi=fnd&pg=PA1&dq=Graph-based+Natural+Language+Processing+and+Information+Retrieval&ots=_EleaeagYQ&sig=zsZwm5PCHToDjARJbtyVhze5q-E&redir_esc=y#v=onepage&q=Graph-based%20Natural%20Language%20Processing%20and%20Information%20Retrieval&f=true)
3. “Automation, Innovation and Work” by Jon-Arild Johannessen, Helene Sætersdal  
[https://books.google.co.in/books?hl=en&lr=&id=jRPWDwAAQBAJ&oi=fnd&pg=PT12&dq=Automation,+Innovation+and+Work+The+Impact+of+Technological,+Economic,+and+Social+Singularity+ByJon-Arild+Johannessen,+Helene+S%C3%A6tersdal&ots=23S8-08S3f&sig=42uQuCVDbaMTuUcRbV6gCZn9mW4&redir\\_esc=y#v=onepage&q=&f=false](https://books.google.co.in/books?hl=en&lr=&id=jRPWDwAAQBAJ&oi=fnd&pg=PT12&dq=Automation,+Innovation+and+Work+The+Impact+of+Technological,+Economic,+and+Social+Singularity+ByJon-Arild+Johannessen,+Helene+S%C3%A6tersdal&ots=23S8-08S3f&sig=42uQuCVDbaMTuUcRbV6gCZn9mW4&redir_esc=y#v=onepage&q=&f=false)
4. “Applied Machine Learning for Smart Data Analysis” by N Dey  
<https://www.taylorfrancis.com/books/edit/10.1201/9780429440953/applied-machine-learning-smart-data-analysis-sanjeev-wagh-parikshit-mahalle-mohd-shafi-pathan-nilanjan-dey>
5. “Adversarial Machine Learning” by D. Joseph  
<https://www.cambridge.org/core/books/adversarial-machine-learning/C42A9D49CBC626DF7B8E54E72974AA3B>
6. “Design of Intelligent Applications using Machine Learning and Deep Learning Techniques” edited by Ramchandra Sharad Mangrulkar  
<https://www.taylorfrancis.com/books/edit/10.1201/9781003133681/design-intelligent-applications-using-machine-learning-deep-learning-techniques-ramchandra-sharad-mangrulkar-antonis-michalas-narendra-shekokar-meera-narvekar-pallavi-vijay-chavan>

7. "Supervised Machine Learning: Optimization Framework and Applications with SAS and R" by Tanya Kolosova and Samuel Berestizhevsky  
<https://www.taylorfrancis.com/books/mono/10.1201/9780429297595/supervised-machine-learning-tanya-kolosova-samuel-berestizhevsky>
8. "Practical Guide to Logistic Regression" by Joseph M. Hilbe  
<https://www.taylorfrancis.com/books/mono/10.1201/b18678/practical-guide-logistic-regression-joseph-hilbe>
9. "Bayesian Modeling and Computation in Python" by Osvaldo A. Martin, Ravin Kumar, Junpeng Lao  
<https://www.taylorfrancis.com/books/mono/10.1201/9781003019169/bayesian-modeling-computation-python-osvaldo-martin-ravin-kumar-junpeng-lao>
10. "Data Science and Analytics with Python" by Jesus Rogel-Salazar  
<https://www.taylorfrancis.com/books/mono/10.1201/9781315151670/data-science-analytics-python-jesus-rogel-salazar>
11. "Chatbot for college website" by Kumar Shivam, Khan Saud, Manav Sharma, Saurav Vashishth, Sheetal Patil  
<https://www.taylorfrancis.com/books/mono/10.1201/9781439834367/software-testing-continuous-quality-improvement-william-lewis>
12. Software Testing and Continuous Quality Improvement by William E. Lewis  
<https://www.taylorfrancis.com/books/mono/10.1201/9780203496329/software-testing-continuous-quality-improvement-gunasekaran-veerapillai-william-lewis>
13. Knowledge Driven Development Bridging Waterfall and Agile Methodologies by Manoj Kumar Lal  
[https://www.researchgate.net/publication/328406837\\_Knowledge\\_Driven\\_Development\\_Bridging\\_Waterfall\\_and\\_Agile\\_Methodologies](https://www.researchgate.net/publication/328406837_Knowledge_Driven_Development_Bridging_Waterfall_and_Agile_Methodologies)
14. "An Analytical Study and Review of open Source Chatbot framework, RASA" by Rakesh Kumar Sharma & Manoj Joshi  
[https://dlwqtxts1xzle7.cloudfront.net/63825165/an-analytical-study-and-review-of-open-IJERTV9IS06072320200704-117508-1fti3xt-libre.pdf?1593859039=&response-content-disposition=inline%3B+filename%3DIJERT\\_An\\_Analytical\\_Study\\_and\\_Review\\_of.pdf&Expires=1685347499&Signature=AohvEDDscbFoi2aUv-qnMgXhz46IXiDIKjux6Q0kOsus3jtUJb7cCXMUTWAq~JeBOm7La-Sx9mEESquQhKqkQ3KVAAJc6QR5VUzj1PbIrl7IFNmwi7tstm-N2Laci8ofX4CjJz9gLus8jueN0Ny0zdtQyrAJOSv1WNdQ2iU7ooumxhLLMj8KYXTBAi4aTQ0Z48d3~9vOlajQrE0XCFT032kFEovsakhCBOujg~VTf5p1G8P~byH9elzVuCJk25UmGgljQnlRbWV~MT6hMpw5sm7g-](https://dlwqtxts1xzle7.cloudfront.net/63825165/an-analytical-study-and-review-of-open-IJERTV9IS06072320200704-117508-1fti3xt-libre.pdf?1593859039=&response-content-disposition=inline%3B+filename%3DIJERT_An_Analytical_Study_and_Review_of.pdf&Expires=1685347499&Signature=AohvEDDscbFoi2aUv-qnMgXhz46IXiDIKjux6Q0kOsus3jtUJb7cCXMUTWAq~JeBOm7La-Sx9mEESquQhKqkQ3KVAAJc6QR5VUzj1PbIrl7IFNmwi7tstm-N2Laci8ofX4CjJz9gLus8jueN0Ny0zdtQyrAJOSv1WNdQ2iU7ooumxhLLMj8KYXTBAi4aTQ0Z48d3~9vOlajQrE0XCFT032kFEovsakhCBOujg~VTf5p1G8P~byH9elzVuCJk25UmGgljQnlRbWV~MT6hMpw5sm7g-)

[EYnsti~bTfe16jmFSsgGr9HITXcSrPjHD86EM29bZ6yr8JncbP1MZe5J0g\\_\\_&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA](#)

15. “College Enquiry Chatbot using Rasa Framework” by Siddhant Meshram, Namit Naik, Megha VR, Tanmay More & Shubhangi Kharche  
<https://ieeexplore.ieee.org/abstract/document/9544650>

### **Web Links**

1. Rasa Chatbot Tutorial | Developed COVID-19 Tracker  
[https://www.youtube.com/watch?v=-JOXupeiIoMU&list=PLIRnO\\_sdVuEevLMSy7bE-Jaqyf1MK\\_wtr](https://www.youtube.com/watch?v=-JOXupeiIoMU&list=PLIRnO_sdVuEevLMSy7bE-Jaqyf1MK_wtr)
2. Your first chatbot with Rasa and Python - Coursera  
<https://www.coursera.org/learn/chatbot-rasa-python/ungradedLti/YEbTy/your-first-chatbot-with-rasa-and-python>
3. Chatbot - Wikipedia  
<https://en.wikipedia.org/wiki/Chatbot>
4. Chatbots Using Python and Rasa  
<https://www.geeksforgeeks.org/chatbots-using-python-and-rasa/>