**TABLE OF CONTENTS**

# ABSTRACT 1. SYSTEM STUDY

* 1. Feasibility Study

1.1.1 Technical Feasibility

1.1.2 Economical Feasibility

1.1.3 Operational Feasibility

1.1.4 Scheduling Feasibility

1.2 System Requirements

1.2.1 Hardware Requirements

1.2.2 Software Requirements

# 2. SYSTEM ANALYSIS

2.1 Introduction

2.2 Project Description

2.3 Modules

2.3.1 Development

2.3.2 Natural Language Understanding (NLU)Model

2.3.3 Implementation

2.3.4 Integration with third party Services

2.3.5 Maintenance and Monitoring

2.3.6 Customer Analytics

2.4 Existing System

2.4.1 Drawbacks of Existing System

2.5 Proposed system

2.5.1 Advantage of Proposed System

# 3. SYSTEM DESIGN

3.1 Data flow Diagram

3.2 Overall Diagram

3.3 Case Diagram

3.4 Class Diagram

3.5 Architecture Diagram

3.6 Source Code

# 4. TESTING

4.1 Unit Testing

4.2 Integration Testing

4.3 Validation Testing

4.4 Acceptance Testing

1. **SYSTEM TESTING & IMPLEMENTATION** 
   1. Introduction
   2. Strategic approach to Software testing
   3. Unit Testing
   4. Software Environment

# 6. RESULTS & CONCLUSIONS

6.1 Results

6.2 Conclusion

6.3 Future Enhancements

# 

**ABSTRACT**

This project aims to create a conversational AI chatbot for a restaurant using the open-source framework, Rasa. The chatbot will provide customers with the ability to order food, book tables, and place takeaway orders through a natural language interface.

In the past, restaurants have relied on traditional methods such as phone calls or in-person visits to place orders and book tables. **This approach has several disadvantages, including long wait times, difficulty tracking orders, and a limited ability to handle high customer requests.**

By implementing a Rasa-based chatbot, restaurants can improve the customer experience by providing a faster and more convenient way to place orders and book tables. The chatbot can be integrated into existing restaurant systems, allowing for seamless order tracking and management.

**CHAPTER -1 SYSTEM STUDY**

### Feasibility Study

The objective of this feasibility study is to determine whether the Rasa-based chatbot restaurant project is viable and achievable within the given constraints. This project is to develop a chatbot for a restaurant that can handle customer inquiries and reservations, provide menu recommendations, and Improve customer satisfaction and engagement through faster and more efficient service Reduce labour costs by automating routine tasks and freeing up staff to focus on more complex customer needs. This feasibility study includes an analysis

* + TECHNICAL FEASIBILITY
  + OPERATIONAL FEASIBILITY
  + ECONOMIC FEASIBILITY
  + SCHEDULING FEASIBILITY

#### Technical Feasibility

**The Rasa open-source framework is a widely used platform for building chatbots, making it technically feasible to develop a chatbot for the restaurant. Additionally, the necessary hardware and software resources are available for the development and implementation of the chatbot.**

#### Economic Feasibility

**The development and implementation of the chatbot require an initial investment of time and resources, including the costs of the Rasa framework and any necessary hardware and software. However, the potential benefits of the chatbot, such as increased efficiency and customer satisfaction, outweigh the initial costs.**

#### Operational Feasibility

**The chatbot project is operationally feasible, as it can improve the restaurant's customer service and streamline its operations. Additionally, the restaurant's existing systems can be integrated with the chatbot, allowing for seamless communication between the chatbot and the staff.**

**1.1.4 Scheduling Feasibility**

**The chatbot project can be completed within The project plan should include timelines for each phase of the project, such as design and planning, development, testing, and implementation.**

### 1.2 System Requirements

#### 1.2.1 Hardware Requirements

Processor: Intel i3 or higher

RAM: 4GB or higher

Hard disk: 500GB or higher

Internet connection: High-speed broadband internet connection

#### 1.2.2 Software Requirements

Operating system: Windowa, Linux, Mac os

Python 3.6 or higher

RASA 2.0 or higher

Visual studio code or any other code editor

## CHAPTER-2 SYSTEM ANALYSIS

### 2.1 Introduction

A chatbot is a computer program designed to simulate conversation with human users, typically over the internet. Chatbots are used in a variety of applications, including customer service, information retrieval, and entertainment. In the context of a restaurant, a chatbot can be used to provide customers with information about menu items, handle orders for food and drink, and assist with booking tables. The chatbot can also be integrated with the restaurant's existing systems, such as its point-of-sale and reservation systems, to streamline the ordering and booking process. This project aims to create a Rasa-powered chatbot for a restaurant that can handle food ordering, table booking, and takeout requests from customers. The chatbot uses natural language processing (NLP) and machine learning algorithms to understand the customer's requests and provide relevant responses. The main advantage of implementing this chatbot is that it provides customers with a convenient and accessible way to interact with the restaurant and place orders. The chatbot is available 24/7, and customers can place orders or book tables at any time without having to wait on hold or speak to a customer service representative. Additionally, the chatbot can handle a large volume of requests simultaneously, reducing the workload on the restaurant's staff and freeing up more time for them to focus on other tasks. Overall, the chatbot has the potential to improve customer satisfaction and increase efficiency in the restaurant.

### 2.2 Project Description

In this development we created a chatbot for a restaurant that uses natural language processing (NLP) to communicate with customers and provide them with information about the restaurant's menu, hours of operation, and other relevant details. The chatbot is built using the Rasa open-source framework and is designed to be integrated with the restaurant's existing systems. The main objectives of the project are to improve the customer experience by providing a quick and convenient way to access information about the restaurant, reduce the workload of restaurant staff by automating repetitive tasks, and gather valuable data on customer preferences and feedback. The chatbot is trained using machine learning algorithms, including intent recognition and entity extraction, to ensure accurate and effective communication with customers. It is capable of answering a wide range of questions and providing personalized recommendations based on customer preferences. The chatbot's integration with the restaurant's existing systems allows it to access and update relevant information in real-time, ensuring that customers receive up-to-date and accurate information at all times. Throughout the project, we worked closely with the team to plan and execute the development, testing, and rollout of the chatbot, ensuring that it met all of the project requirements and was user-friendly and effective. Overall, our chatbot project provides a valuable tool for restaurants to improve their customer experience, reduce staff workload, and gather valuable data on customer preferences and feedback.

### Modules

#### Development

#### **The process of creating the chatbot, including designing and planning its features and functionality, selecting the appropriate technology stack, and building the software using programming languages, frameworks, and libraries.**

#### Natural Language Understanding (NLU) Training

Natural Language Understanding (NLU) Training is a technique used to teach the chatbot how to understand and interpret human language by using machine learning algorithms to analysis and extract meaning from text data. This involves creating and fine-tuning the chatbot's language models, entity recognition, and intent classification

.

#### 2.3.3 Implementation

The process of deploying the chatbot into a production environment, which includes setting up the necessary infrastructure, integrating it with other systems, and configuring it to work as intended.

#### Integration with third-party services

The process of connecting the chatbot to external services or APIs, such as payment gateways, ordering systems, customer relationship management (CRM) software, etc., to enhance its functionality and streamline customer interactions.

* + 1. **Maintenance and monitoring**

Ongoing support and upkeep of the chatbot after its deployment, including troubleshooting, bug fixes, updates, and performance monitoring to ensure optimal functionality and customer satisfaction

* + 1. **Customer Analytics**

The process of collecting, analyzing, and interpreting data on customer interactions with the chatbot, such as usage patterns, feedback, sentiment analysis, and other metrics to gain insights and improve the chatbot's performance

### Existing System

### In the Existing system I, a chatbot has been created specifically for the university utilizing the Myanmar language. This chatbot aims to provide a convenient and accessible platform for students, staff, and faculty to gather information and receive assistance. The chatbot leverages advanced Natural Language Processing techniques to understand the queries and provide accurate and relevant responses. By incorporating the Myanmar language, the chatbot caters to the local language preferences and enhances the overall user experience. Additionally, the chatbot is available 24/7, making it accessible to users at any time and from any location. With this innovative technology, the university is working towards providing a more efficient and effective means of communication for its community

#### 2.4.1 Drawbacks of Existing System

* Limited audience reach
* Language complexity
* Translation issues
* User adoption

### 2.5 Proposed system

The proposed system for this project is a Rasa-powered chatbot for a restaurant. The chatbot is designed to provide customers with a convenient and accessible way to interact with the restaurant and place orders. The chatbot uses NLP and machine learning algorithms to understand the customer's requests and provide relevant responses, reducing the workload on the restaurant's staff and improving efficiency. The system consists of the following components:

Natural Language Understanding (NLU) Component: This component is responsible for understanding the customer's requests and extracting relevant information, such as the type of food they want to order, the date and time they want to book a table, etc.

Dialogue Management Component: This component is responsible for guiding the conversation and determining the appropriate response for the customer's request. It uses machine learning algorithms to make predictions about the customer's intent and the relevant information to respond with.

Action Server: This component implements custom actions, such as ordering food or booking tables, that the chatbot can perform in response to customer requests. The action server is responsible for executing these actions and providing the relevant responses to the customer.

User Interface: This component provides the interface for customers to interact with the chatbot. This could be in the form of a chatbot interface on a restaurant website or a messaging platform like Facebook Messenger.

Backend Systems: This component represents the restaurant's existing systems, such as its point-of-sale and reservation systems. The chatbot is integrated with these systems to streamline the ordering and booking process.

**2.5.1 Advantage of Proposed System**

* Increased efficiency
* 24/7 availability
* Improved customer experience
* Accurate and consistent
* Cost-effective

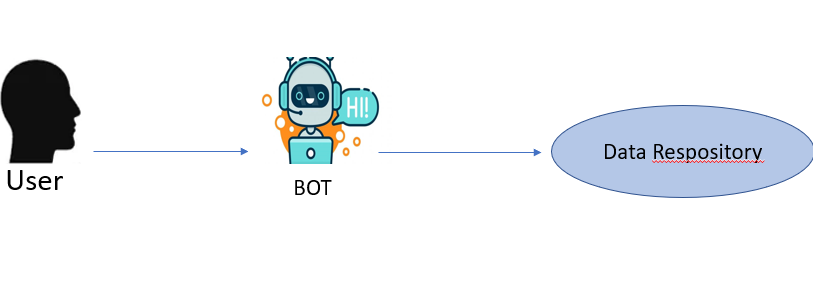
**CHAPTER - 3**

**SYSTEM DESIGN**

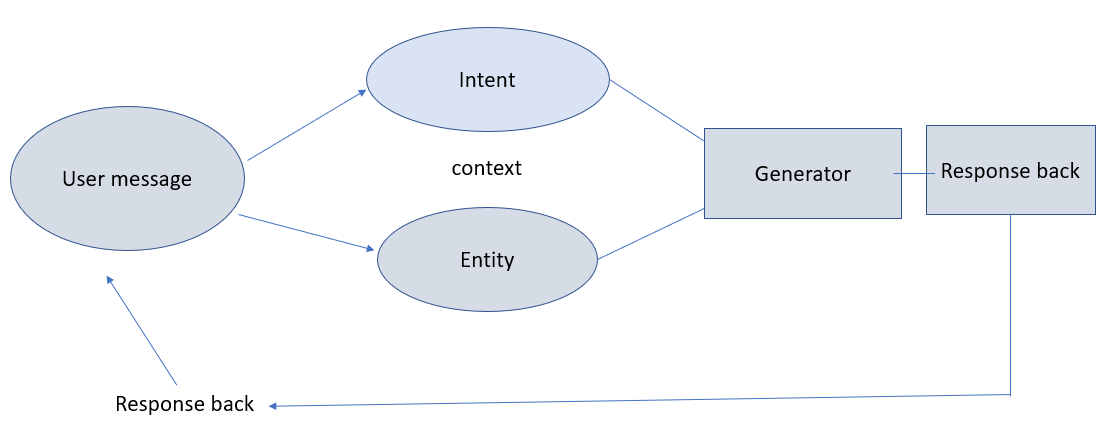
**3.1 Data flow Diagram**

# LEVEL-0

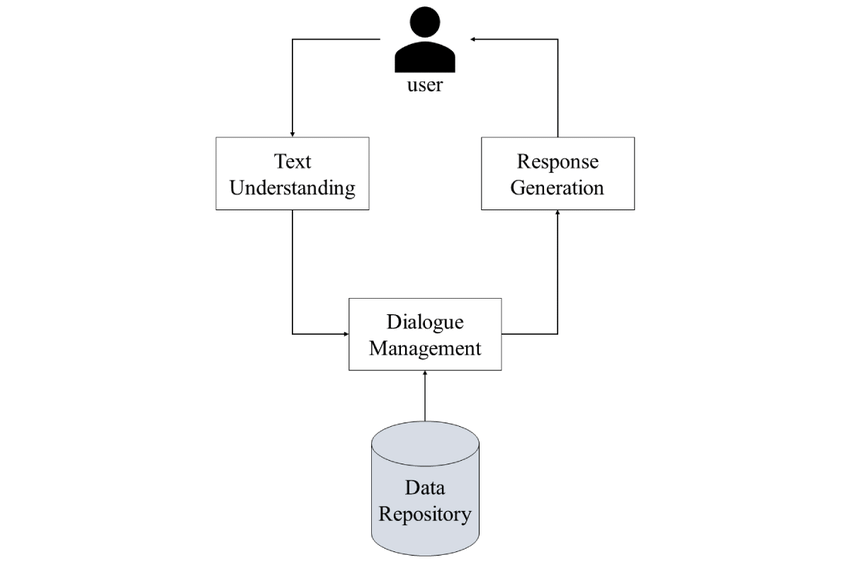
# 



**LEVEL-1**

****

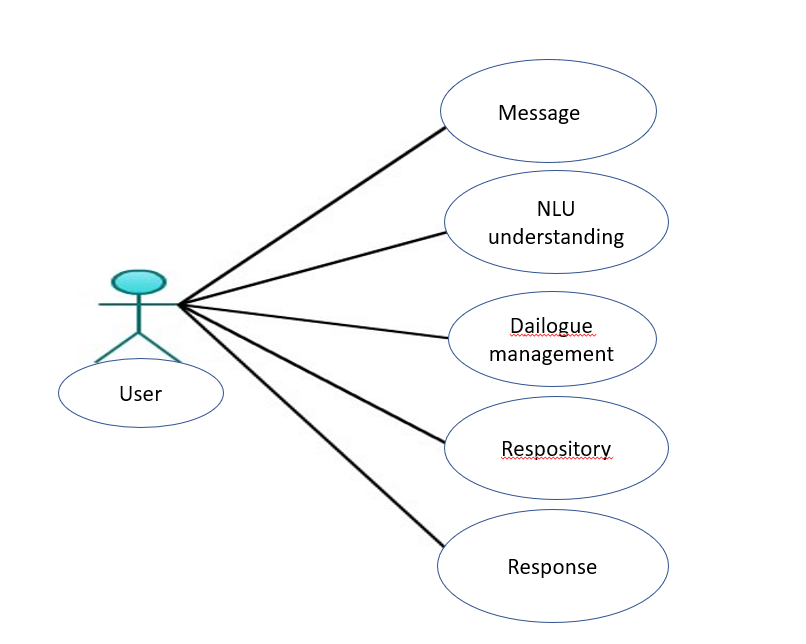
**LEVEL-2**



## 3.2 Overall Diagram

## 

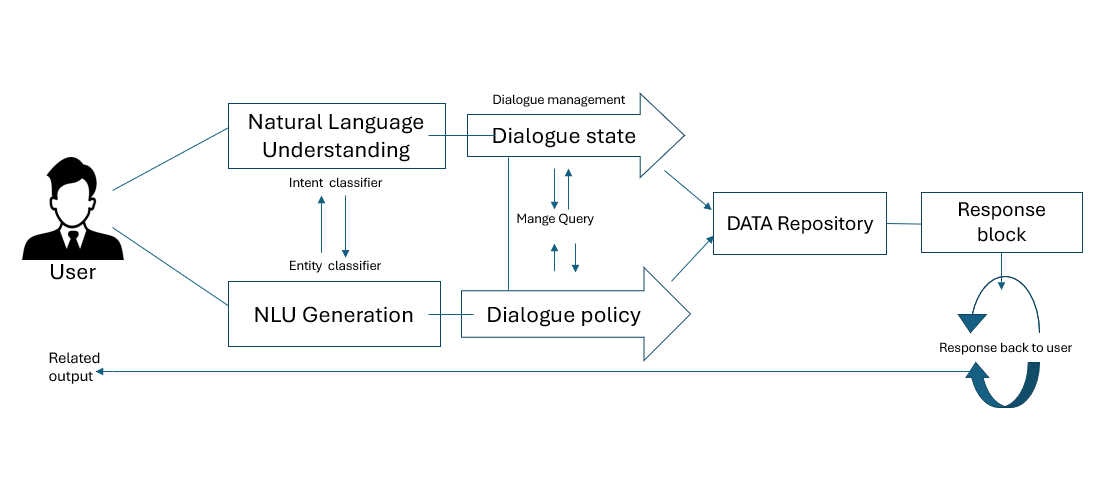
## 3.3 Case Diagram



## 3.4 Class Diagram

****

## 3.5 Architecture Diagram



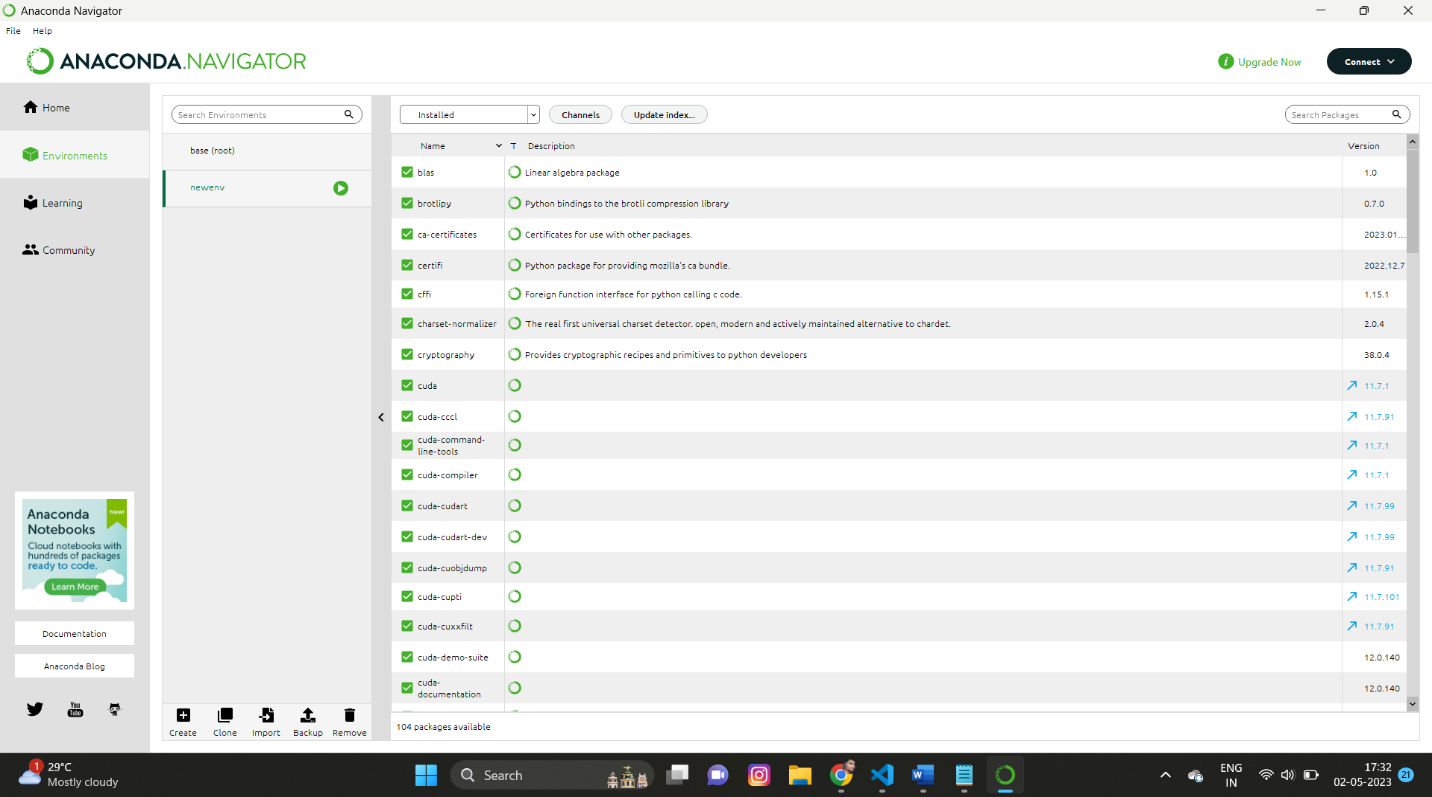
**3.6 Source Code CODING:**

**Step 1: Installation Instructions for Rasa Open Source and Anaconda Environment**

1. Install Rasa Open-Source framework: The first step in setting up Rasa is to install the Rasa Open-Source framework. Rasa provides comprehensive installation instructions for all major operating systems, including Windows, macOS, and Linux.
2. Install Anaconda: Next, you need to install Anaconda, which is a popular data science platform that includes a package manager, an environment manager, and other tools for data analysis and machine learning.
3. Create a virtual environment: Once Anaconda is installed, create a new virtual environment for your Rasa project using the Anaconda command-line interface.
4. Install Rasa dependencies: Install the necessary dependencies for Rasa, such as TensorFlow and spa Cy, in your virtual environment using the pip package manager.
5. Create a new Rasa project: Finally, create a new Rasa project using the Rasa command-line interface. This will generate the necessary files and folders for your chatbot project, including the NLU data and the dialogue management model.

**Step 2: To set up an environment in Anaconda:**

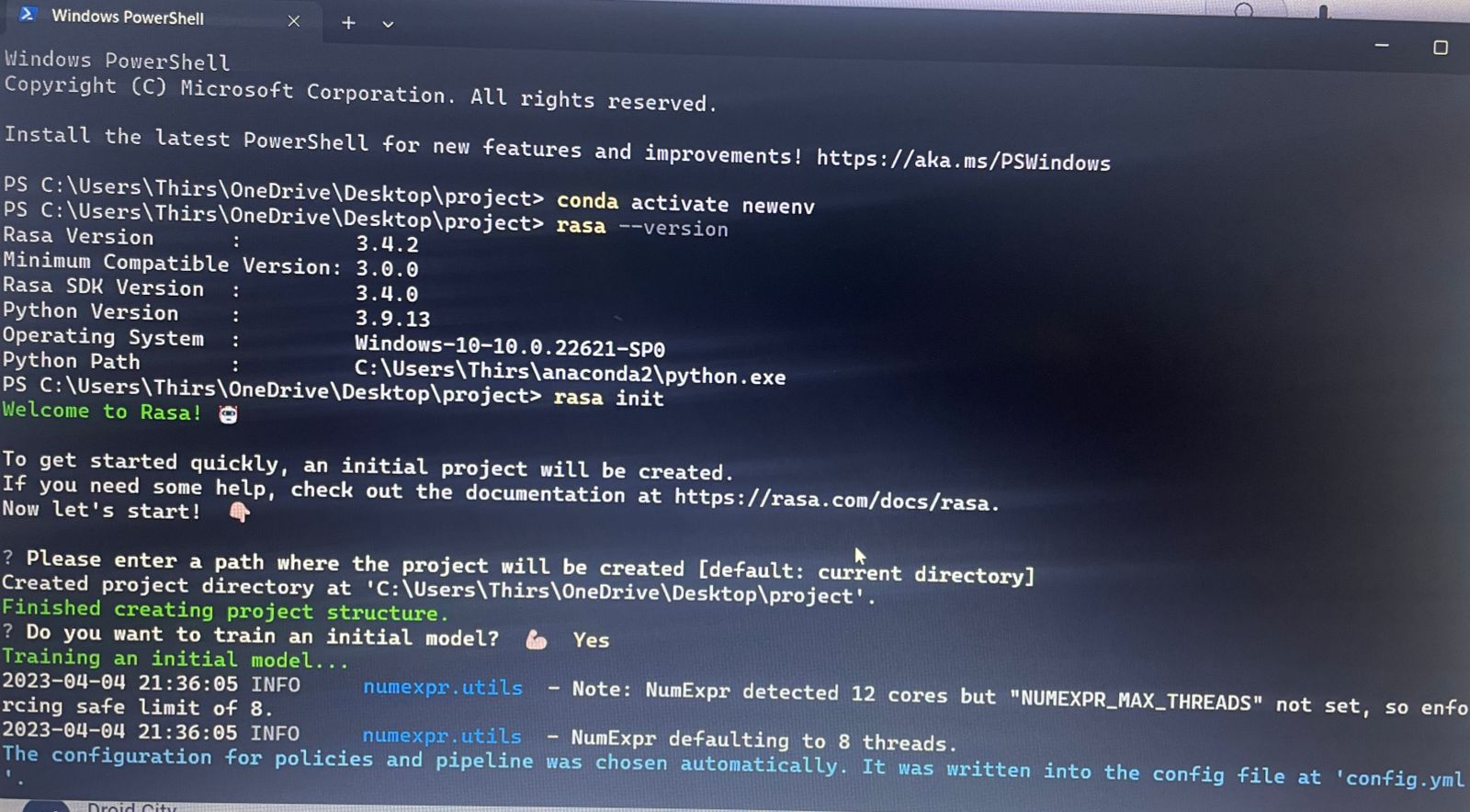
1. Open the Anaconda Navigator.
2. Click on the "Environments" tab on the left-hand side of the window.
3. Click on the "Create" button on the bottom of the window.
4. In the "Create Environment" dialog box, give a name to your new environment.
5. Select the Python version you want to use.
6. Choose the packages you want to install in your new environment. You can select from the list of available packages or search for a specific package.
7. Click on the "Create" button to create the environment.
8. Once the environment is created, you can activate it by clicking on the play button on the right-hand side of the window.

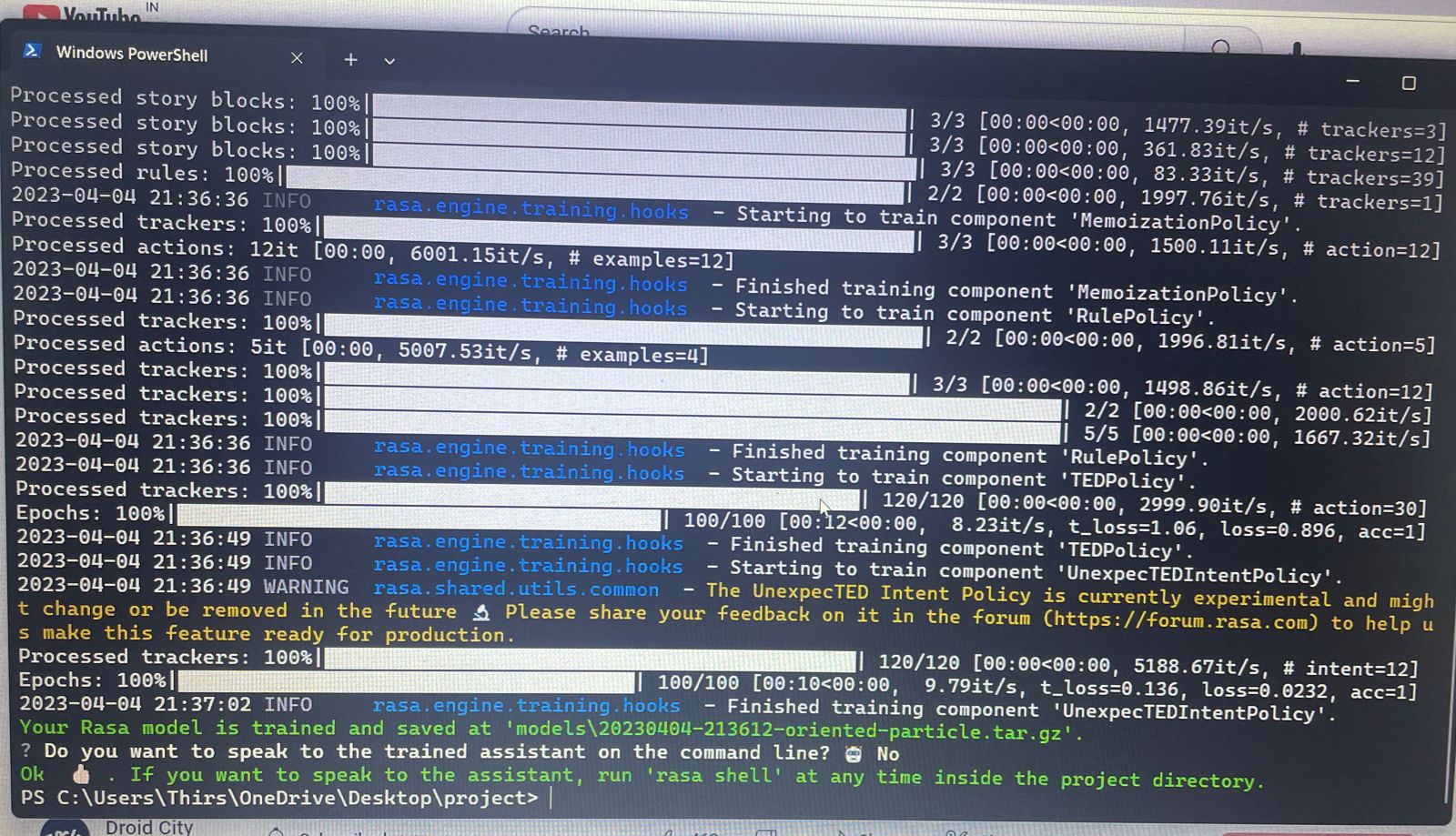


**Step 3: Activate Rasa Environment in Anaconda**

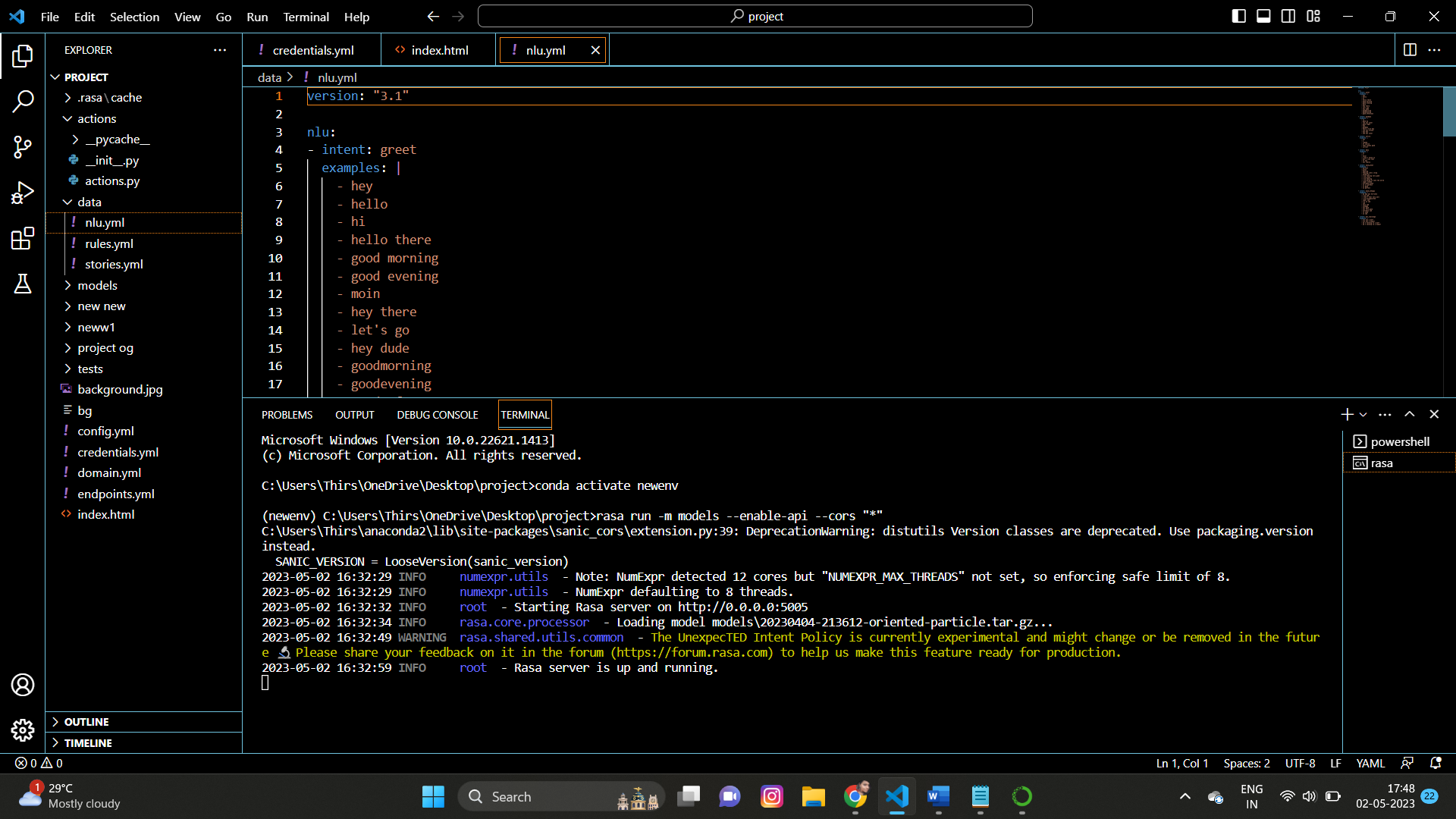
Once you have installed Anaconda, the next step is to activate the Rasa environment. To do this, you need to open the Anaconda Prompt and run the following command: **conda activate name (**the name which you have created for the Anaconda environment **) Example: conda activate newenv(**new environment)

This command will activate the Rasa environment and prepare it for use. You will see the name of the environment in the command prompt, indicating that you are now working within the Rasa environment. From here, you can install Rasa and any other necessary packages or libraries to begin building your chatbot.





* Once the Rasa package is installed, navigate to the folder where it is installed.
* Open the folder in Visual Studio Code or any other preferred code editor.



**To Train the NLU Model in Rasa :**

**The NLU data**

**#** Intent for placing an order

- intent: order\_food

examples: |

- I want to order pizza

- Can I get a hamburger and fries?

- Could you bring me a glass of water?

- Can we have a Caesar salad to start?

- I would like to order the spaghetti carbonara please

- Can I see the menu?

# Intent for asking about menu items

- intent: ask\_menu

examples: |

- What do you have on the menu?

- Could you tell me about your vegetarian options?

- Is the seafood fresh?

- What's the daily special?

- Do you have gluten-free dishes?

- How spicy is the chicken curry?

# Intent for making a reservation

- intent: make\_reservation

examples: |

- I want to make a reservation for tonight

- Can I book a table for four people?

- Do you have any availability for tomorrow?

- What time slots do you have for dinner?

- Can we reserve the private room?

- I would like to make a reservation for a birthday party

# Intent for asking about restaurant information

- intent: ask\_restaurant\_info

examples: |

- What are your opening hours?

- Where are you located?

- Do you have parking?

- What's your phone number?

- Is there a dress code?

- Are you pet-friendly

**stories.yml**

**## Story 1**

# **\* greet**

# **- utter\_greet**

# **\* ask\_menu**

# **- utter\_menu**

# **## Story 2**

# **\* greet**

# **- utter\_greet**

# **\* order\_food{"food\_item": "pizza"}**

# **- slot{"food\_item": "pizza"}**

# **- action\_order\_food**

# **- utter\_confirmation**

# **## Story 3**

# **\* greet**

# **- utter\_greet**

# **\* make\_reservation{"date": "tonight", "party\_size": "4"}**

# **- slot{"date": "tonight", "party\_size": "4"}**

# **- action\_make\_reservation**

# **- utter\_reservation\_confirmation**

# **## Story 4**

# **\* greet**

# **- utter\_greet**

# **\* ask\_restaurant\_info{"information\_type": "location"}**

# **- slot{"information\_type": "location"}**

# **- action\_provide\_restaurant\_info**

# **- utter\_thanks/**

version: '2.0'

rules:

- rule: Order food

steps:

- intent: order\_food

- action: utter\_ask\_order\_details

- intent: inform

- action: utter\_confirm\_order

- rule: Make reservation

steps:

- intent: make\_reservation

- action: utter\_ask\_reservation\_details

- intent: inform

- action: utter\_confirm\_reservation

- rule: Ask about menu

steps:

- intent: ask\_menu

- action: utter\_menu\_options

- rule: Ask about restaurant info

steps:

- intent: ask\_restaurant\_info

- action: utter\_restaurant\_info

Action code :

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 []

**config.yml file**

pipeline:

# - name: WhitespaceTokenizer

# - name: RegexFeaturizer

# - name: LexicalSyntacticFeaturizer

# - name: CountVectorsFeaturizer

# - name: CountVectorsFeaturizer

# analyzer: 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

# Configuration for Rasa Core.

# https://rasa.com/docs/rasa/core/policies/

policies:

# - name: MemoizationPolicy

# - name: RulePolicy

# - name: UnexpecTEDIntentPolicy

# max\_history: 5

# epochs: 100

# - name: TEDPolicy

# max\_history: 5

# epochs: 100

# constrain\_similarities**: true**

**Credentials file**

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"

**Main Domain Code :**

intents:

- order\_food

- ask\_menu

- make\_reservation

- ask\_restaurant\_info

- greet

- goodbye

- affirm

- deny

- inform

- thank\_you

slots:

cuisine:

type: unfeaturized

num\_people:

type: unfeaturized

seating\_type:

type: unfeaturized

requested\_reservation\_time:

type: unfeaturized

requested\_reservation\_date:

type: unfeaturized

phone\_number:

type: unfeaturized

location:

type: unfeaturized

budget:

type: unfeaturized

entities:

- cuisine

- num\_people

- seating\_type

- requested\_reservation\_time

- requested\_reservation\_date

- phone\_number

- location

- budget

responses:

utter\_greet:

- text: "Hello! How can I assist you?"

utter\_goodbye:

- text: "Goodbye! Have a nice day."

utter\_ask\_cuisine:

- text: "What type of cuisine are you interested in?"

utter\_ask\_num\_people:

- text: "How many people are in your party?"

utter\_ask\_seating\_type:

- text: "Would you prefer indoor or outdoor seating?"

utter\_ask\_reservation\_time:

- text: "What time would you like to reserve a table for?"

utter\_ask\_reservation\_date:

- text: "What date would you like to reserve a table for?"

utter\_ask\_phone\_number:

- text: "May I have your phone number to confirm the reservation?"

utter\_ask\_location:

- text: "Where are you located?"

utter\_ask\_budget:

- text: "Do you have a budget in mind?"

actions:

- utter\_greet

- utter\_goodbye

- utter\_ask\_cuisine

- utter\_ask\_num\_people

- utter\_ask\_seating\_type

- utter\_ask\_reservation\_time

- utter\_ask\_reservation\_date

- utter\_ask\_phone\_number

- utter\_ask\_location

- utter\_ask\_budget

slots:

cuisine:

type: unfeaturized

num\_people:

type: unfeaturized

seating\_type:

type: unfeaturized

requested\_reservation\_time:

type: unfeaturized

requested\_reservation\_date:

type: unfeaturized

phone\_number:

type: unfeaturized

location:

type: unfeaturized

budget:

type: unfeaturized

intents:

- order\_food

- ask\_menu

- make\_reservation

- ask\_restaurant\_info

- greet

- goodbye

- affirm

- deny

- inform

- thank\_you

entities:

- cuisine

- num\_people

- seating\_type

- requested\_reservation\_time

- requested\_reservation\_date

- phone\_number

- location

- budget

slots:

cuisine:

type: unfeaturized

num\_people:

type: unfeaturized

seating\_type:

type: unfeaturized

requested\_reservation\_time:

type: unfeaturized

requested\_reservation\_date:

type: unfeaturized

phone\_number:

type: unfeaturized

location:

type: unfeaturized

budget:

type: unfeaturized

templates:

Mongo DB tracker

tracker\_store:

type: mongod

url: mongodb://localhost:27017

db: chatbot\_tracker

action\_endpoint:

url: <http://localhost:5055/webhook>

Rasa domain

from typing import Any, Text, Dict, List

from rasa\_sdk import Action, Tracker

from rasa\_sdk.executor import CollectingDispatcher

from rasa\_sdk.forms import FormAction

from rasa\_sdk.events import SlotSet

class RestaurantForm(FormAction):

"""Restaurant reservation form"""

def name(self) -> Text:

return "restaurant\_form"

@staticmethod

def required\_slots(tracker: Tracker) -> List[Text]:

return ["guests", "date", "time", "cuisine", "location"]

def slot\_mappings(self) -> Dict[Text, Any]:

return {"guests": self.from\_entity(entity="guests",

intent=["inform\_reservation"]),

"date": self.from\_entity(entity="date",

intent=["inform\_reservation"]),

"time": self.from\_entity(entity="time",

intent=["inform\_reservation"]),

"cuisine": self.from\_entity(entity="cuisine",

intent=["inform\_cuisine"]),

"location": self.from\_entity(entity="location",

intent=["inform\_location"])}

def submit(self,

dispatcher: CollectingDispatcher,

tracker: Tracker,

domain: Dict[Text, Any]) -> List[Dict]:

# Save the reservation details in MongoDB

guests = tracker.get\_slot("guests")

date = tracker.get\_slot("date")

time = tracker.get\_slot("time")

cuisine = tracker.get\_slot("cuisine")

location = tracker.get\_slot("location")

# Code for saving data to MongoDB

return []

class ActionGreet(Action):

"""Greet the user"""

def name(self) -> Text:

return "action\_greet"

def run(self,

dispatcher: CollectingDispatcher,

tracker: Tracker,

domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

dispatcher.utter\_message(text="Hello! How can I help you today?")

return []

class ActionGoodbye(Action):

"""Say goodbye to the user"""

def name(self) -> Text:

return "action\_goodbye"

def run(self,

dispatcher: CollectingDispatcher,

tracker: Tracker,

domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

dispatcher.utter\_message(text="Goodbye! Have a nice day.")

return []

class ActionDefaultFallback(Action):

"""Default fallback action"""

def name(self) -> Text:

return "action\_default\_fallback"

def run(self,

dispatcher: CollectingDispatcher,

tracker: Tracker,

domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

dispatcher.utter\_message(text="Sorry, I didn't understand. Can you please rephrase?")

return []

class ActionUnhandled(Action):

"""Handle unhandled intents"""

def name(self) -> Text:

return "action\_unhandled"

def run(self,

dispatcher: CollectingDispatcher,

tracker: Tracker,

domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

dispatcher.utter\_message(text="I'm sorry, I don't know how to help with that. Can I assist you with something else?")

return []

class ActionRestart(Action):

"""Restart the conversation"""

def name(self) -> Text:

return "action\_restart"

def run(self,

dispatcher: CollectingDispatcher,

tracker: Tracker,

domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

return [SlotSet(slot, None) for slot in self.slots\_to\_reset]

**HTML file**

<html>

<body>

<!-- <img src="./background.jpg" alt="" style="height:100% "> -->

<script>!(function () {

let e = document.createElement("script"),

t = document.head || document.getElementsByTagName("head")[0];

(e.src =

"https://cdn.jsdelivr.net/npm/rasa-webchat@1.x.x/lib/index.js"),

(e.async = !0),

(e.onload = () => {

window.WebChat.default(

{

initPayload: '/greet',

customData: { language: "en" },

socketUrl: "http://loqcalhost:5005",

// add other props here

},

null

);

}),

t.insertBefore(e, t.firstChild);

})();

</script>

</body>

</html>

**Website domain code :**

<!DOCTYPE html>

<html lang="en">

<head>

<title>The Venue</title>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="description" content="The Venue template project">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" type="text/css" href="styles/bootstrap-4.1.2/bootstrap.min.css">

<link href="plugins/font-awesome-4.7.0/css/font-awesome.min.css" rel="stylesheet" type="text/css">

<link rel="stylesheet" type="text/css" href="plugins/OwlCarousel2-2.2.1/owl.carousel.css">

<link rel="stylesheet" type="text/css" href="plugins/OwlCarousel2-2.2.1/owl.theme.default.css">

<link rel="stylesheet" type="text/css" href="plugins/OwlCarousel2-2.2.1/animate.css">

<link href="plugins/colorbox/colorbox.css" rel="stylesheet" type="text/css">

<link href="plugins/jquery-datepicker/jquery-ui.css" rel="stylesheet" type="text/css">

<link href="plugins/jquery-timepicker/jquery.timepicker.css" rel="stylesheet" type="text/css">

<link rel="stylesheet" type="text/css" href="styles/main\_styles.css">

<link rel="stylesheet" type="text/css" href="styles/responsive.css">

</head>

<body>

<!DOCTYPE html>

<html>

<head>

<title>Chatbot Example</title>

<style>

#chatbot {

position: fixed;

bottom: 0;

right: 0;

width: 300px;

height: 300px;

margin-right: 25px;

background-color: #fff;

border: 1px solid #ccc;

box-shadow: 0 0 10px rgba(0,0,0,0.3);

z-index: 999;

font-family: Arial, sans-serif;

}

#chatbot form {

display: flex;

margin: 0;

padding: 10px;

border-top: 1px solid #ccc;

}

#chatbot input[type="text"] {

flex: 1;

padding: 5px;

font-size: 16px;

border: none;

border-radius: 3px;

}

#chatbot button[type="submit"] {

/\* padding: 5px 10px; \*/

font-size: 16px;

border: none;

border-radius: 5px;

background-color: #007bff;

color: #fff;

cursor: pointer;

}

#chatbot #chatLog {

list-style: none;

margin: 0;

padding: 10px;

height: 100%;

overflow-y: auto;

}

#chatbot .botMessage {

background-color: #f1f0f0;

border-radius: 3px;

padding: 5px;

margin-bottom: 10px;

}

#chatbot .userMessage {

background-color: #007bff;

color: #fff;

border-radius: 3px;

padding: 5px;

margin-bottom: 10px;

}

</style>

</head>

<body>

<div id="chatbot">

<form>

<input type="text" id="userInput" placeholder="Type your message...">

<button type="submit" id="submitBtn">Send</button>

</form>

<ul id="chatLog">

</ul>

</div>

<script>

// Define the bot's response

const botResponse = 'Hello, how can I help you today?';

// Get references to the form and chat log

const form = document.querySelector('#chatbot form');

const chatLog = document.querySelector('#chatbot #chatLog');

// Handle form submission

form.addEventListener('submit', function(event) {

// Prevent the form from submitting and refreshing the page

event.preventDefault();

// Get the user input

const userInput = document.querySelector('#userInput').value.trim();

// Clear the user input

document.querySelector('#userInput').value = '';

// Add the user's message to the chat log

const userMessage = document.createElement('li');

userMessage.className = 'userMessage';

userMessage.textContent = userInput;

chatLog.appendChild(userMessage);

// Add the bot's message to the chat log

const botMessage = document.createElement('li');

botMessage.className = 'botMessage';

botMessage.textContent = botResponse;

chatLog.appendChild(botMessage);

// Scroll to the bottom of the chat log

chatLog.scrollTop = chatLog.scrollHeight;

});

</script>

</body>

</html>

<script>

// Define the bot's response

const botResponse = 'Hello, how can I help you today?';

// Get references to the form and chat log

const form = document.querySelector('#chatbot form');

const chatLog = document.querySelector('#chatbot #chatLog');

// Handle form submission

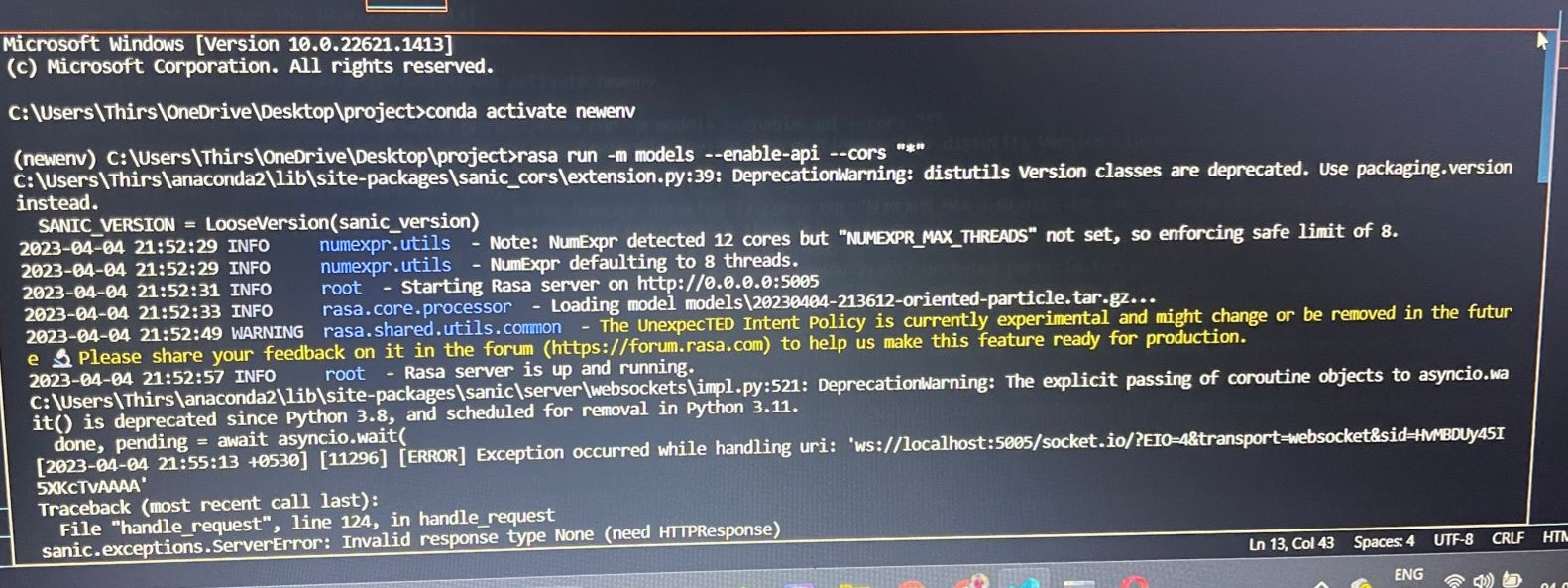
form.addEventListener('submit', function(event) {

// Prevent the form from submitting and refreshing the page

event.preventDefault();

// Get the user input

Running commands are :



## CHAPTER-4

## TESTING

### 4.1 Unit Testing

In the development of the chatbot for the restaurant, a comprehensive unit testing strategy was employed to ensure the functionality and accuracy of the system. All individual functions and modules were tested independently to identify any errors or bugs in the coding. The chatbot's natural language processing models were also tested using a variety of inputs to ensure accurate and relevant responses. Various test cases were designed to simulate different user scenarios, such as placing an order or asking for menu recommendations. These test cases were executed multiple times to ensure the chatbot's consistency and reliability. Any errors or bugs identified during testing were documented and addressed through debugging and code optimization. Overall, the unit testing phase played a critical role in ensuring the accuracy and effectiveness of the chatbot in communicating with customers

### 4.2 Integration Testing

Integration testing involves testing the communication and interaction between different components of a system to ensure that they function properly together. In the case of the chatbot for the restaurant, integration testing was conducted to verify that the chatbot was able to communicate with third-party services such as payment gateways, ordering systems, and customer databases. This testing was done by simulating user interactions with the chatbot and ensuring that the data was accurately transmitted and processed by all the components of the system. Any errors or inconsistencies were identified and resolved during this testing phase..

### 4.3 Validation Testing

process of ensuring that the chatbot meets the specified requirements and functions correctly according to the user's needs. It involves verifying that the chatbot is user-friendly, efficient, and provides accurate responses to user queries. The validation testing process includes creating test scenarios, identifying expected results, and performing tests to verify the functionality of the chatbot. Through validation testing, any discrepancies or issues in the chatbot can be identified and resolved, ensuring that the chatbot meets the expected standards of quality and user satisfaction.

### 4.4 Acceptance Testing

In this testing phase, a group of representative users interact with the chatbot and evaluate its performance based on predefined acceptance criteria. The acceptance criteria includes factors such as accuracy of responses, speed of response, ability to handle various user inputs and scenarios, and overall user experience. During the acceptance testing phase, any defects or issues found are documented and reported to the development team for resolution. Once the chatbot passes the acceptance testing, it is considered ready for deployment and use by the restaurant and its customers.

**CHAPTER - 5**

## SYSTEM TESTING & IMPLEMENTATION

### 5.1 Introduction

Software testing is the process of evaluating a software product to identify any discrepancies between the expected and actual outcomes, ensuring that the software meets the specified requirements and works as expected. The objective of software testing is to uncover any defects or bugs before the software is released to the end-users, ensuring that the software is of high quality and meets the customer's needs. Software testing involves a range of techniques and tools, including manual and automated testing, to ensure that the software is reliable, efficient, and scalable. The importance of software testing lies in its ability to identify and correct errors early in the software development cycle, minimizing the risks associated with software failures, and ultimately improving the quality of the software product.

### 5.2 Strategic approach to Software testing

Requirement Analysis: This involves analysing the requirements of the chatbot and identifying the key features and functionalities it should possess.

1. **Develop test scenarios and cases:** Create a comprehensive list of test scenarios and cases that cover all the functionality of the chatbot, including both positive and negative scenarios.
2. **Defect Tracking and Reporting:** Any issues found during testing should be tracked and reported, and appropriate action should be taken to resolve them.
3. **Prioritize test cases:** Prioritize test cases based on their criticality and impact on the overall system.
4. **Conduct compatibility testing:** Verify that the chatbot works seamlessly across different platforms and devices.

**5. Perform regression testing:** Repeat the testing process after each software update or change to ensure that the chatbot's performance and functionality have not been affected.

**6**. **Test Closure:** Once all tests are executed, the results are analyzed, and the chatbot is deemed ready for release, the testing process is closed.

### 5.3 Unit Testing

During the unit testing phase of the chatbot project, several types of tests were performed on the individual functions and modules of the system. The primary goal of unit testing was to identify and correct errors in the code and ensure that each component of the the chatbot was functioning as expected The various types of tests conducted during unit testing included Input/output testing, Functional testing, Integration testing, Performance testing, Security testing

### 1. Input/output Testing

Input testing is a type of testing that checks how the chatbot handles different types of inputs, such as text, images, and voice commands. It ensures that the chatbot is able to recognize and process various inputs accurately and appropriately. This type of testing also involves checking for errors or unexpected behaviour when invalid or ambiguous inputs are provided to the chatbot.

Output testing, on the other hand, checks the responses and outputs generated by the chatbot in response to various inputs. This includes testing the accuracy and relevancy of the responses, as well as the format and structure of the output. Output testing also involves checking for errors or unexpected behaviour in the chatbot's responses, such as repetition or irrelevant information.

### 2. Functional Testing

Functional testing for a chatbot involves testing the bot's ability to perform its intended functions, such as responding to user queries, providing accurate information, and completing tasks. Some common types of functional testing for chatbots include:

1. **Intent testing:** Verifying that the chatbot correctly understands and interprets user intents.

2. **Entity testing:** Verifying that the chatbot correctly identifies and extracts relevant entities from user input.

3. **Flow testing:** Testing the flow of conversation between the user and the chatbot to ensure it is smooth and seamless.

4. **Error handling testing**: Testing the chatbot's ability to handle unexpected inputs or errors gracefully.

5**. Regression testing:** Testing the chatbot after changes or updates to ensure that previously working functions still work correctly.

**3. Integration Testing**

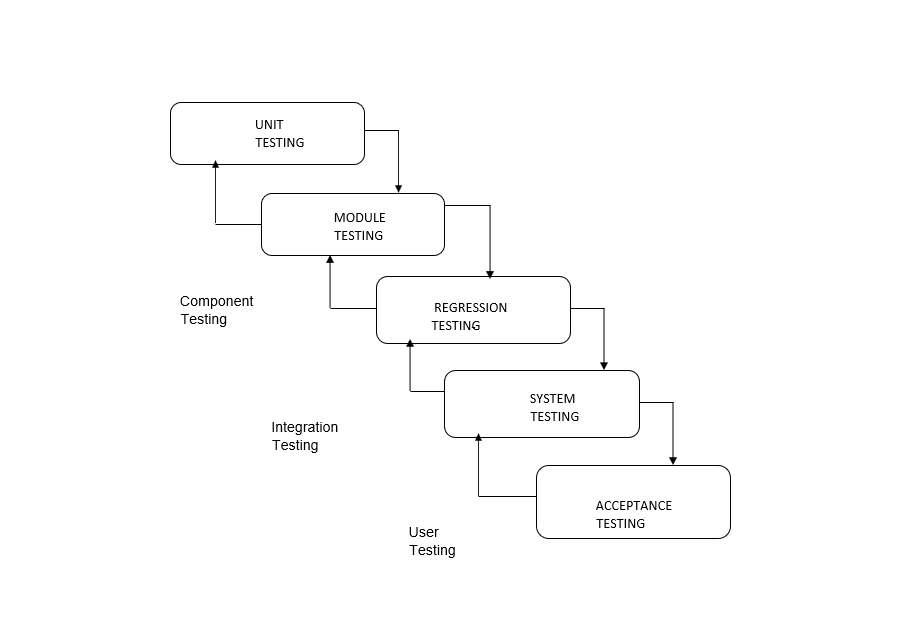
During integration testing, different modules and components of the chatbot are brought together and tested as a whole. This includes testing the user interface, the Natural Language Processing (NLP) engine, and the integration with external systems. Integration testing can be performed at different levels, including unit testing, system testing, and acceptance testing.

### 4. Performance Testing

This testing is typically conducted to evaluate the chatbot's ability to handle a high volume of user requests and interactions without slowing down or crashing. Performance testing involves simulating various user scenarios and measuring the chatbot's response time, throughput, and resource utilization under different load conditions. The goal is to ensure that the chatbot is scalable and can handle the expected number of users without compromising its performance or user experience. This type of testing is critical for identifying and resolving any performance bottlenecks or issues before the chatbot is deployed in a production environment.

### 5. Security Testing

The goal of security testing is to identify any potential threats or breaches that could compromise the confidentiality, integrity, and availability of data within the chatbot system. This type of testing involves a range of techniques, including vulnerability scanning, penetration testing, and security audits. Security testing for chatbots also involves analysing the security of any third-party integrations that the chatbot may have with other systems, such as payment gateways or ordering systems. By conducting thorough security testing, restaurant owners can ensure that their chatbot system is secure and can protect customer data from potential threats or breaches.



**5.4 Software Environment**

**1.Python Technology**

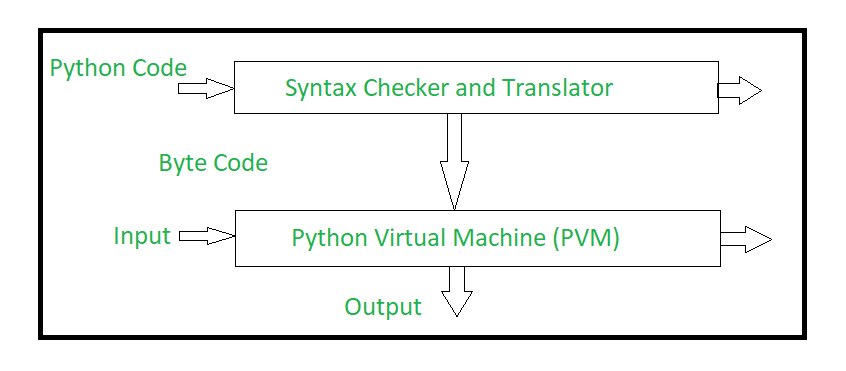
Python is a high-level, interpreted programming language that is widely used for general-purpose programming and it is an object-oriented programming language

### The Python Programming Language

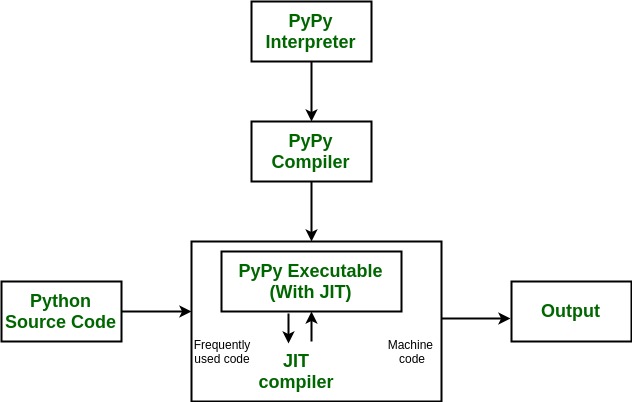
Python is a high-level programming language that emphasizes code readability and simplicity. It is characterized by its

* Clear sytax
* dynamic typing
* high-level data structures
* Object oriented
* Ease of use
* simplicity
* High performance
* Interpreted
* Multithreaded
* Secure

Python is often used for scripting, automation, and building web applications, as well as for machine learning and artificial intelligence. It also has a large standard library that includes modules for a variety of tasks such as web development, scientific computing, and data analysis.



It uses dynamic typing and automatic memory management, making it easy to learn and use. Python supports a wide range of programming paradigms, including procedural, object-oriented, and functional programming. It has a large standard library and a vibrant community of developers who contribute to third-party libraries and frameworks. Python is commonly used in web development, scientific computing, data analysis, artificial intelligence, and many other domains. It is known for its clear syntax, ease of use, and wide range of applications.



**The High-level programming language**

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of their features support functional programming and aspect-oriented programming. Many other paradigms are supported via extensions, including design by contract and logic programming Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management. It uses dynamic name resolution (late binding), which binds method and variable names during program execution. Its design offers some support for functional programming in the Lisp tradition. It has a filter, map and reduce functions; list comprehensions, dictionaries, sets, and generator expressions The standard library has two modules (tools and functions) that implement functional tools borrowed from Haskell and Standard ML.

Python has several components, including:

**1.Standard Library:** Python comes with a large and comprehensive standard library that provides a wide range of functionalities for tasks like file handling, regular expressions, networking, and more.

**2.Interpreter:** Python is an interpreted language, which means it doesn't need to be compiled like languages such as C++ or Java. The Python interpreter executes the code line by line, which makes the development process faster.

**3.Integrated Development Environments (IDEs):** Python has several popular IDEs, such as PyCharm, Spyder, and Jupyter Notebook, that provide a range of features like code debugging, syntax highlighting, code completion, and more to make development easier

**4.Third-Party Libraries:** Python has a vast collection of third-party libraries and modules that can be used for various purposes, such as data analysis, scientific computing, web development, and more.

**5.Extensibility:** Python can be easily extended by writing code in other languages like C or C++, which can be compiled into a shared library and then imported into Python. This makes it possible to leverage existing libraries and functionality from other languages within Python.

**Development environments**

Python also comes with an Integrated development environment (IDE) called IDLE, which is more beginner-oriented. Other shells, including IDLE and IPython, add further abilities such as improved auto-completion, session state retention, and syntax highlighting. As well as standard desktop integrated development environments, there are Web browser-based IDEs, including SageMath, for developing science- and math-related programs; PythonAnywhere, a browser-based IDE and hosting environment; and Canopy IDE, a commercial IDE emphasizing scientific computing

**Where is Python used ?**

Python is a general-purpose, popular programming language and it is used in almost every technical field. The various areas of Python use are given below.

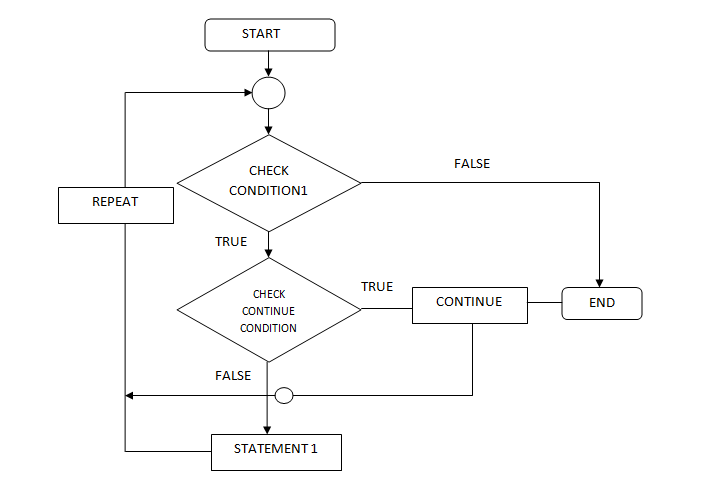
* Data Science
* Date Mining
* Desktop Applications
* Console-based Applications
* Mobile Applications
* Software Development
* Artificial Intelligence
* Web Applications
* Enterprise Applications
* 3D CAD Applications
* Machine Learning
* Computer Vision or Image Processing Applications.
* Speech Recognitions
* AD

**What is python used for ?**

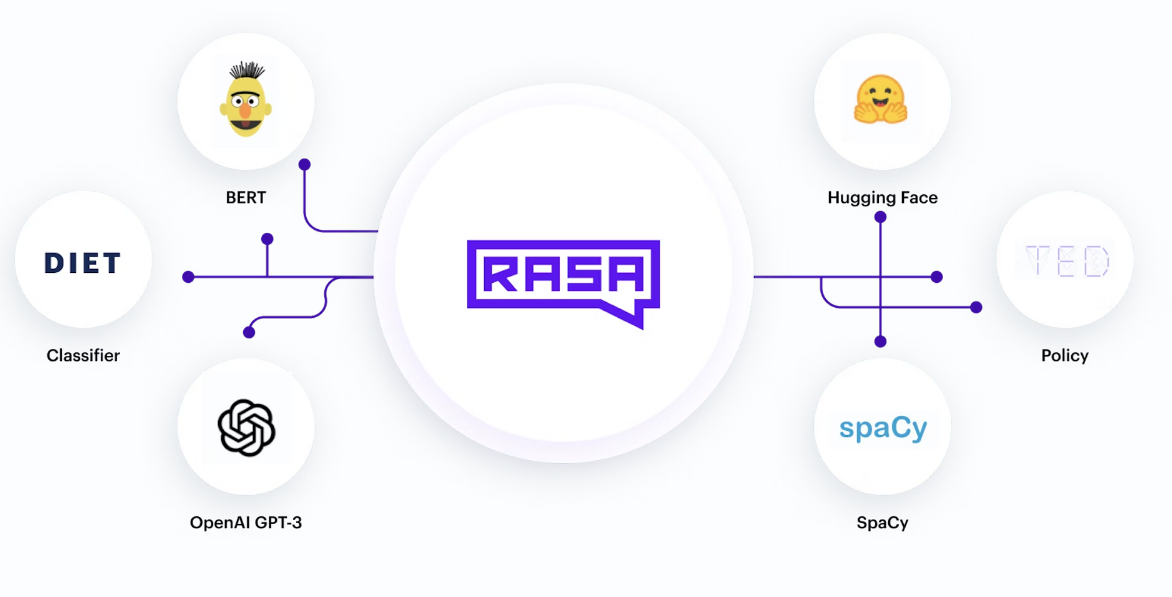
Python is commonly used for developing websites and software, task automation, data analysis, and data visualization. Since it’s relatively easy to learn, Python has been adopted by many non-programmers such as accountants and scientists, for a variety of everyday tasks, like organizing finances.

* Data analysis and machine learning
* Web development
* Automation or scripting
* Software testing and prototyping

**Flow control**

**2.Rasa framework:**

Rasa is an open-source machine learning framework for building AI assistants and chatbots. Mostly you don’t need any programming language experience to work in Rasa. Although there is something called “Rasa Action Server” where you need to write code in Python, that mainly used to trigger External actions like Calling Google API or REST API etc



**Rasa has two main modules:**

* Rasa NLU for understanding user messages
* Rasa Core for holding conversations and deciding what to do next

Rasa X is a tool that helps you build, improve, and deploy AI Assistants that are powered by the Rasa framework. Rasa X includes a user interface and a REST API. Rasa X is the latest release from Rasa.

**Rasa components:**

* Rasa X — It’s a Browser based GUI tool which will allow you to train Machine learning model by using GUI based interactive mode. Remember it’s an optional tool in Rasa Software Stack. Sometimes Rasa sends usage statistics information from your browser to rasa — but it never sends training data to outside of your system, it just sends how many times you are using Rasa X Train.
* Rasa NLU — This is the place, where rasa tries to understand User messages to detect Intent and Entity in your message. Rasa NLU has different components for recognizing intents and entities, most of which have some additional dependencies.
* Spacy (You need to install it separately)
* Tensorflow (By Default available with Rasa)
* Rasa Core — This is the place, where Rasa try to help you with contextual message flow. Based on User message, it can predict dialogue as a reply and can trigger Rasa Action Server.

Rasa internally uses Tensorflow, whenever you do “pip install rasa” or “pip install rasa-x”, by default it installs Tensorflow.

Rasa is the essential platform for creating superior customer experiences and Rasa provides flexible conversational AI software for building text and voice-based assistants. Used by developers, conversational teams, and enterprises worldwide.

It Keep user conversations completely confidential, and protect your IP. Rasa allows you to run your assistant's operations on your own infrastructure, without sending customer messages to a hosted, third party service for processing. It Deploy on-premises or on your own private cloud. Rasa deploys on your own infrastructure, in even the strictest enterprise IT environments. Flexible architecture that lets you control access to data.

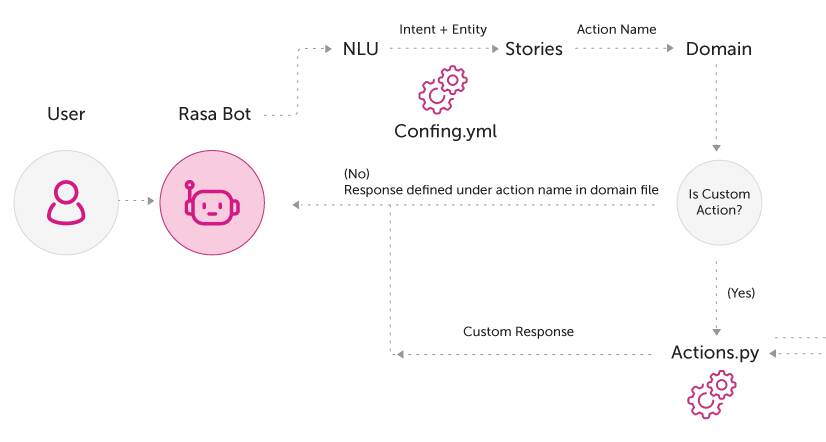
**Benefits of creating a chatbot using Rasa, including:**

* Open-source: Rasa is an open-source framework, meaning that it is free to use and there is a large community of developers contributing to its development and improvement.
* Customizable: Rasa provides a high level of customization, allowing developers to build a chatbot tailored to their specific needs.
* Natural Language Processing (NLP): Rasa has strong NLP capabilities, allowing it to understand and interpret natural language queries with a high degree of accuracy.
* Scalability: Rasa is designed to be scalable, allowing chatbots to handle large volumes of requests with ease.
* Integration: Rasa can be integrated with a wide range of other tools and platforms, including messaging apps, voice assistants, and customer service software.
* Analytics: Rasa provides detailed analytics and insights into chatbot performance, allowing developers to continually improve and refine their chatbot over time.

**Rasa work process steps:**

* Defining the domain
* Writing stories
* Training the NLU
* Training the Dialogue Management (DM)
* Testing the chatbot
* Deploying the chatbot

**Rasa chatbot workflow**

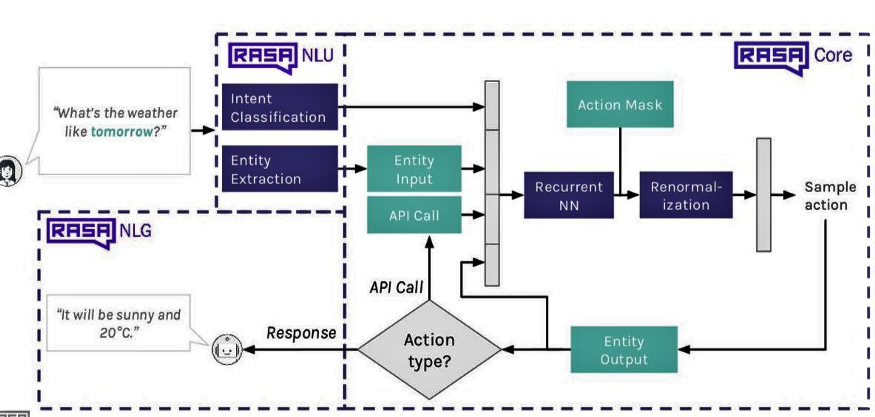


**3.Rasa Natural Language Understanding Module** (NLU)

Rasa NLU (Natural Language Understanding) is an open-source library for building natural language processing models. It allows developers to build and improve chatbots, virtual assistants, and other conversational interfaces by training machine learning models to understand user inputs in natural language. Rasa NLU is an open-source natural language processing (NLP) tool that enables developers to build custom conversational agents with high accuracy and flexibility. Rasa NLU uses machine learning algorithms to analyze and understand natural language inputs from users, allowing chatbots to provide accurate and relevant responses. The tool can be trained on specific language models and can be customized to tailored to their specific needs. With Rasa NLU, developers have full control over the chatbot's responses, as the tool can be integrated with custom business logic and third-party APIs.

**NLU use case**

The primary use of Rasa NLU is to extract relevant information from user inputs and map them to intents and entities. Intents represent the purpose or goal of the user's message, while entities represent important pieces of information within the message. For example, in a restaurant chatbot, the intent could be to make a reservation, and the entities could be the date, time, and number of people



**Benefits of using Rasa NLU**

* Customization: Rasa NLU provides a high degree of customization, allowing developers to create models that fit their specific use cases and language domains.
* Open-source: Rasa NLU is open-source, meaning that it is free to use, modify, and distribute.
* Machine learning: Rasa NLU uses machine learning algorithms to improve over time, meaning that it can learn from user interactions and become more accurate.
* Integration with Rasa Core: Rasa NLU integrates seamlessly with Rasa Core, a dialogue management framework, to create intelligent chatbots and virtual assistants.

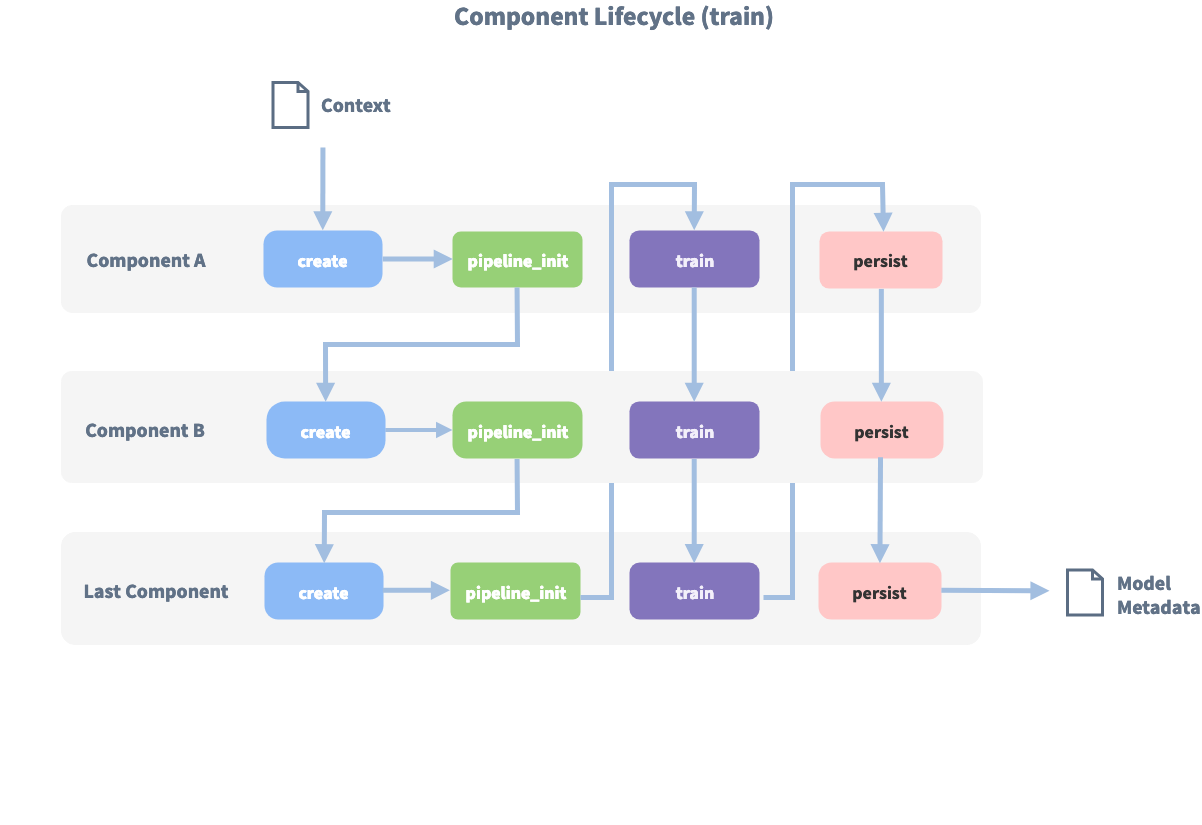
Overall, Rasa NLU provides a powerful tool for building natural language understanding capabilities into chatbots and other conversational interfaces, enabling more intuitive and efficient interactions between humans and machines.

**Rasa NLU process**

The Rasa NLU (Natural Language Understanding) process involves the following steps:

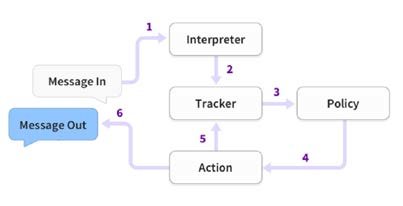
* **Data collection and pre-processing:** The first step is to collect and pre process the training data. The training data usually consists of user messages and their corresponding intents and entities. Pre processing involves cleaning the data, removing irrelevant information, and converting the data into a usable format for training the model.
* **Training the model:** Once the data is collected and pre processed, the next step is to train the NLU model. Rasa NLU provides various algorithms for training the model, such as spacy \_sklearn, tensorflow\_ embedding, and mitie. During training, the model learns to recognize user intents and extract entities from user messages.
* **Model evaluation:** After the model is trained, it is evaluated to measure its performance. The evaluation is done by testing the model on a separate set of data and measuring its accuracy, precision, recall, and F1 score.
* **Model tuning:** Based on the evaluation results, the model is tuned to improve its performance. Tuning involves adjusting the hyperparameters of the model, such as the number of epochs, learning rate, and batch size.
* **Integration with chatbot platform:** Finally, the trained and tuned model is integrated with the chatbot platform to enable it to understand user messages and respond accordingly. The chatbot platform communicates with the NLU model to get the user intent and entities, which are then used to trigger the appropriate action or response.

Overall, the Rasa NLU process plays a crucial role in the development of chatbots by enabling them to understand user messages and provide relevant responses.



# 4.Rasa core:

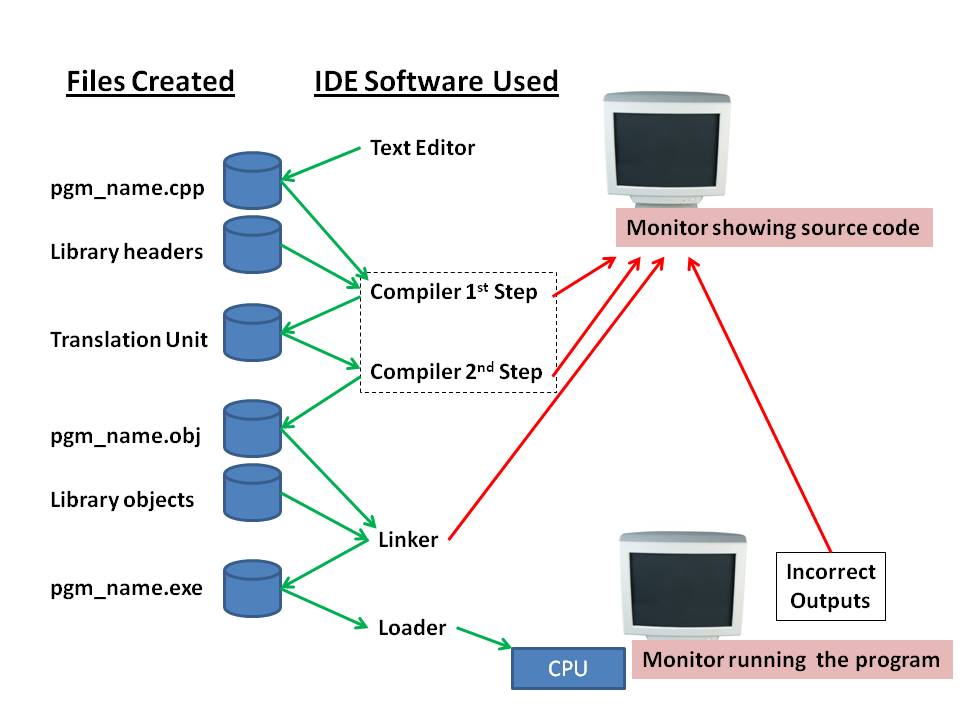
# This is the module used for dialogue management and to handle the conversation flow. Rasa Core is a machine learning-based dialogue management system that allows chatbots to hold more engaging and natural conversations with users. It is used to build intelligent assistants that can understand user intent, keep track of context, and handle complex dialogue flows. Some common use cases for Rasa Core include customer service chatbots, personal assistants, and chatbots for e-commerce platforms. By using Rasa Core, these chatbots can provide a more seamless user experience by handling complex interactions and providing more personalized responses. For example, an e-commerce chatbot built with Rasa Core could help a user navigate through a variety of product options, recommend products based on the user's preferences, and even help complete a purchase. The benefits of using Rasa Core include its flexibility and scalability. It allows developers to define custom actions and behaviours for their chatbots and can handle complex conversations with ease. Rasa Core can also integrate with various natural language processing tools, including Rasa NLU, to further improve the accuracy and relevance of its responses. Additionally, Rasa Core can be deployed in various environments, including on-premise and in the cloud, making it a versatile and adaptable choice for building chatbots.



## 5. Integrated Development Environment(IDE)

An Integrated Development Environment such as PyCharm or Visual Studio Code can be used for coding and testing the bot. An Integrated Development Environment (IDE) is a software application that provides a comprehensive environment for developing software. It typically includes a code editor, compiler/interpreter, debugger, and other tools necessary for software development. IDE are designed to increase the productivity and efficiency of developers by providing them with a centralized platform for developing, testing, and debugging code. They often include features such as code highlighting, code completion, version control integration, and automated testing.

One of the main benefits of using an IDE is that it can help streamline the development process by reducing the amount of time and effort required to perform common tasks such as debugging and code testing. Additionally, IDEs can improve the overall quality of the code being developed by providing tools for code analysis and refactoring. There are many IDEs available for different programming languages and platforms. Some of the most popular IDEs include Visual Studio, Eclipse, IntelliJ IDEA, and PyCharm. Each IDE has its own strengths and weaknesses, and developers often choose an IDE based on their specific needs and preferences.



## 6.MongoDB

This is a NoSQL database used to store user information and chat history. MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License (SSPL) which is deemed non-free by several distributions. MongoDB is a member of the MACH Alliance.

**Main features**

1. **Ad-hoc queries:** MongoDB supports field, range query, and regular-expression searches. Queries can return specific fields of documents and also include user-defined JavaScript functions. Queries can also be configured to return a random sample of results of a given size.
2. **Indexing:** Fields in a MongoDB document can be indexed with primary and secondary indices or indexes.
3. **Replication:** MongoDB provides high availability with replica sets. A replica set consists of two or more copies of the data. Each replica-set member may act in the role of the primary or secondary replica at any time. All writes and reads are done on the primary replica by default. Secondary replicas maintain a copy of the data of the primary using built-in replication. When a primary replica fails, the replica set automatically conducts an election process to determine which secondary should become the primary. Secondaries can optionally serve read operations, but that data is only eventually consistent by default.
4. **Load balancing:** MongoDB scales horizontally using sharding. The user chooses a shard key, which determines how the data in a collection will be distributed. The data is split into ranges (based on the shard key) and distributed across multiple shards. (A shard is a master with one or more replicas.) Alternatively, the shard key can be hashed to map to a shard – enabling an even data distribution.
5. **File storage:** MongoDB can be used as a file system, called GridFS, with load balancing and data replication features over multiple machines for storing files.
6. **Aggregation:** MongoDB provides three ways to perform aggregation: the aggregation pipeline, the map-reduce function, and single-purpose aggregation methods. Map-reduce can be used for batch processing of data and aggregation operations. But according to MongoDB's documentation, the Aggregation Pipeline provides better performance for most aggregation operations.
7. **Server-side JavaScript execution:** JavaScript can be used in queries, and aggregation functions (such as MapReduce), and sent directly to the database to be executed.
8. **Capped collections:** MongoDB supports fixed-size collections called capped collections. This type of collection maintains insertion order and, once the specified size has been reached, behaves like a circular queue.
9. **Transactions:** MongoDB claims to support multi-document ACID transactions. This claim was found to not be true as MongoDB violates snapshot isolation.

## Architecture

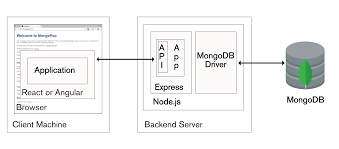
* Programming language accessibility
* MongoDB has official drivers for major programming languages and development environments. There are also a large number of unofficial or community-supported drivers for other programming languages and frameworks.
* Serverless access
* Management and graphical front-ends
* Record insertion in MongoDB with Robo mongo
* The primary interface to the database has been the mongo shell. MongoDB Compass is introduced as the native GUI. There are products and third-party projects that offer user interfaces for administration and data viewing.

## Benefits of MongoDB

* Flexible data model: MongoDB has a flexible schema, which allows you to store data in a variety of formats and structures, without the need to define a rigid schema upfront.
* Scalability: MongoDB is designed to scale horizontally, allowing you to distribute data across multiple servers and handle large volumes of data and traffic.
* High availability: MongoDB includes features like replica sets and sharding, which provide automatic failover and ensure high availability and data redundancy.
* Performance: MongoDB is optimized for fast reads and writes, with support for indexes and other performance-enhancing features.
* Rich query language: MongoDB includes a powerful query language with support for complex queries, geospatial queries, text search, and aggregation.
* Document-oriented: MongoDB stores data in JSON-like documents, which allows you to store rich, hierarchical data structures in a single document.
* Open-source: MongoDB is open-source software, which means it is free to use and has a large and active community of users and contributors.
* Cloud-native: MongoDB is built for the cloud, with support for modern deployment architectures like containerization, microservices, and serverless computing.

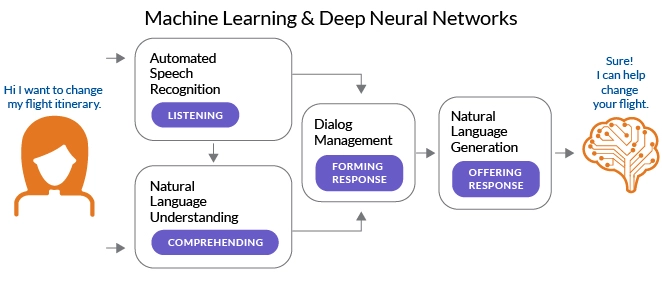
## ****MongoDB aid in the creation of a chatbot for restaurants****

MongoDB can be used as a database to store information related to the restaurant, such as menus, customer orders, and user profiles. The chatbot can use MongoDB to store and retrieve this information as needed to provide personalized and accurate responses to users. Additionally, MongoDB's flexible document model allows for easy and efficient data modeling and management, making it well-suited for handling unstructured data, such as user inputs and chatbot responses. This makes it easier to develop and maintain the chatbot's conversation flow and improves its accuracy over time. MongoDB's scalability and performance capabilities also enable the chatbot to handle a large volume of user requests and respond quickly, providing a seamless user experience. It can also integrate with other services and tools to further enhance the chatbot's functionality, such as natural language processing and machine learning frameworks.



## 7. Conversational AI Platform

Conversational AI platforms are software tools that enable the development and deployment of chatbots and virtual assistants that can interact with humans in a natural and intuitive way. These platforms use various artificial intelligence techniques, including natural language processing (NLP), machine learning, and deep learning, to understand and respond to human input in a conversational manner. Conversational AI platforms often include pre-built components such as dialogue management, intent recognition, and entity extraction, which can be customized and extended to meet the specific needs of a particular chatbot or virtual assistant application. The ultimate goal of conversational AI platforms is to enable businesses and organizations to provide better customer service, automate tasks, and enhance user engagement through natural and human-like conversations. Conversation AI platforms typically offer a range of tools and features for building, training, and deploying chatbots and virtual assistants. These platforms often use natural language processing (NLP) and machine learning (ML) techniques to enable more sophisticated and human-like interactions. Some conversation AI platforms are designed specifically for certain industries or use cases, such as customer service, healthcare, or e-commerce. Many platforms also provide analytics and reporting features to help businesses understand how their chatbots are performing and identify areas for improvement. Some conversation AI platforms offer pre-built templates and integrations with other tools and systems, making it easier and faster to get started with building a chatbot.

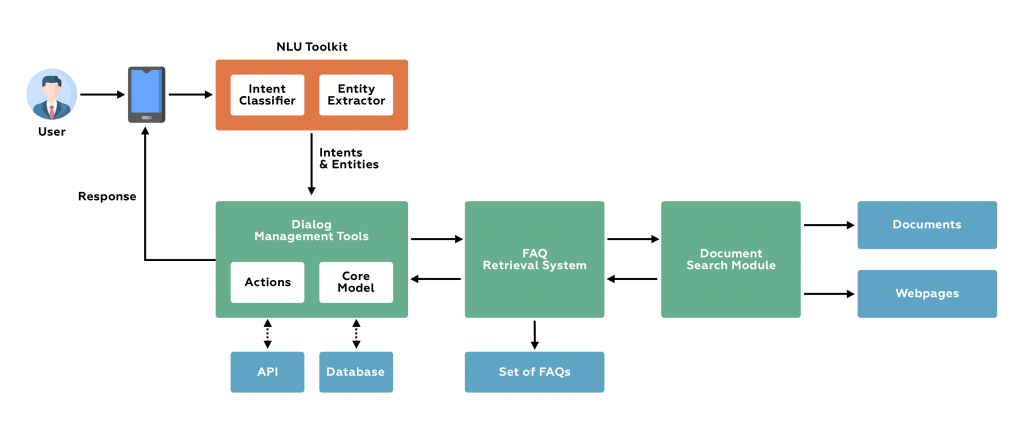


**conversational AI be used to create a chatbot for a restaurant**

Conversational AI is used to create chatbots for restaurants by providing a natural language interface for customers to interact with. This interface allows the customers to ask questions, place orders, make reservations, and receive personalized recommendations in a conversational manner. The chatbot is built using machine learning algorithms that enable it to understand natural language input, recognize customer intents, and provide appropriate responses.

Conversational AI is also used to enhance the customer experience by providing 24/7 support, reducing wait times, and improving the accuracy of orders. It can also be integrated with other systems, such as payment gateways, to provide a seamless and secure experience for customers. Additionally, conversational AI can be used to collect feedback and data from customers, which can be analyzed to improve the quality of service and make data-driven decisions for the restaurant.

Overall, conversational AI can be a powerful tool for restaurants to improve customer engagement, increase revenue, and streamline operations.



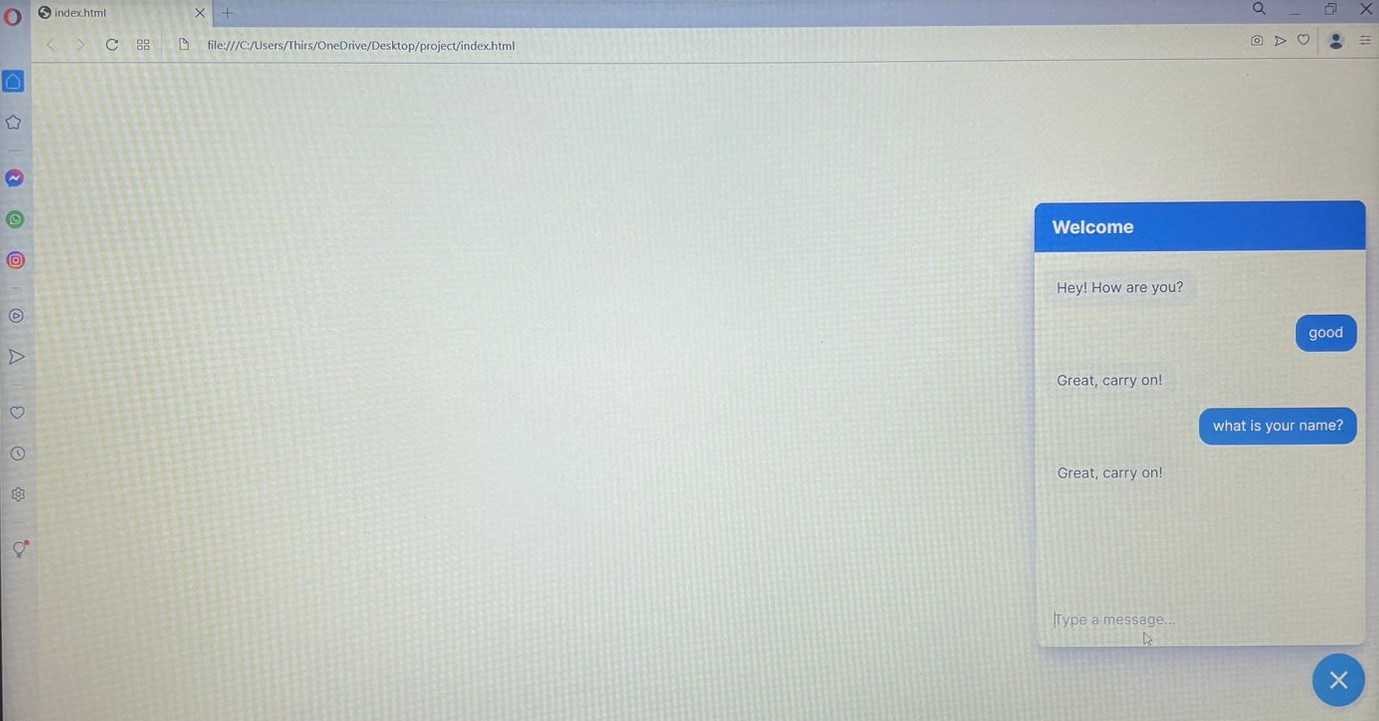
**Benefits of using AI in chatbots, including:**

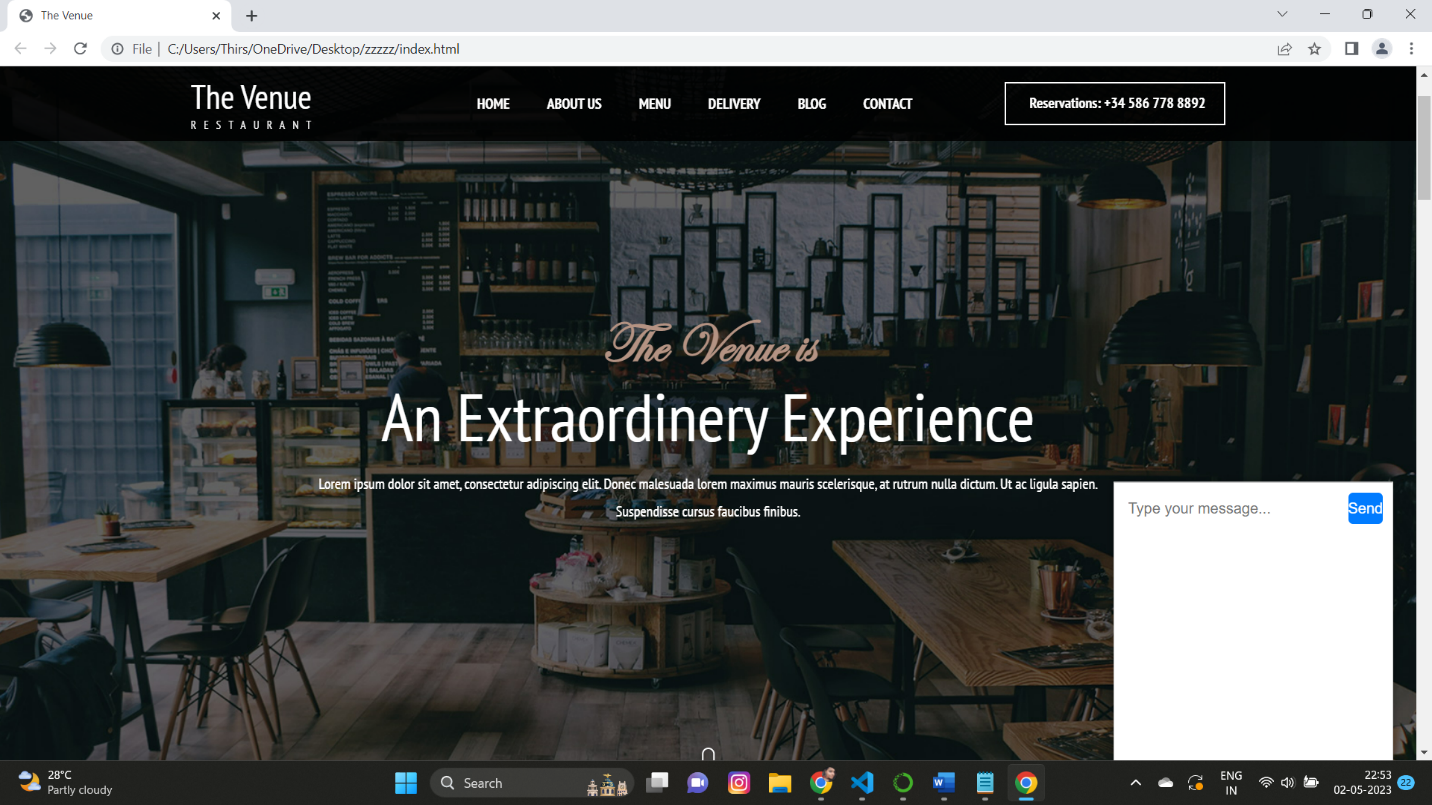
* **Personalization:** AI can be used to analyze user data and provide personalized experiences, including customized recommendations and responses.
* **Automation:** Chatbots powered by AI can automate tasks such as answering frequently asked questions, scheduling appointments, and processing orders, which can save time and increase efficiency.
* **Improved customer service:** AI-powered chatbots can provide 24/7 support, reduce wait times, and provide quick and accurate responses to customer inquiries.
* **Cost savings:** By automating tasks and reducing the need for human support, chatbots powered by AI can help reduce operational costs.
* **Scalability:** Chatbots can handle multiple customer inquiries simultaneously, allowing businesses to scale their customer support without increasing staffing costs.
* **Data analysis:** AI-powered chatbots can gather and analyze customer data, providing businesses with valuable insights into customer behaviour and preferences.

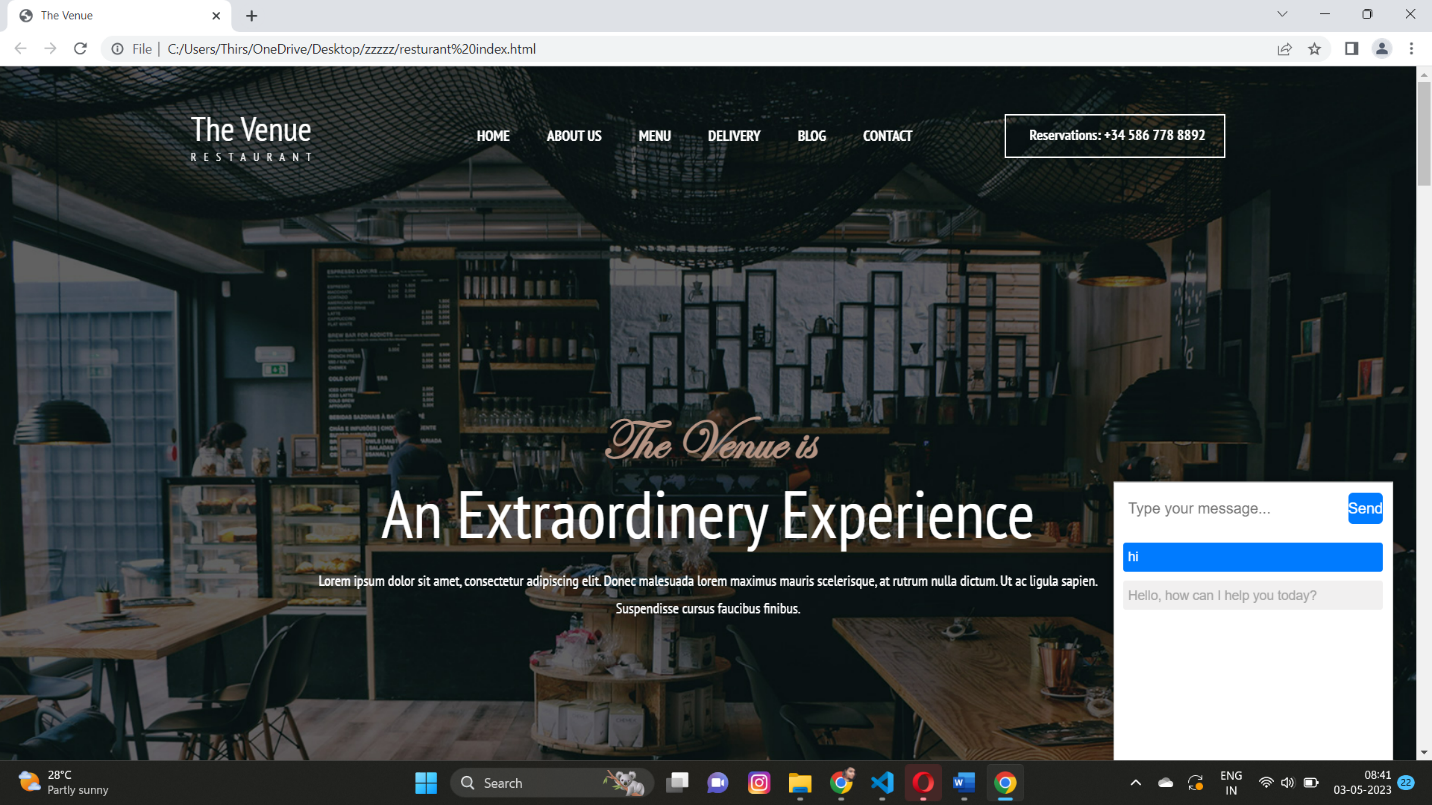
**CHAPTER – 6**

**RESULTS & CONCLUSION**

### 6.1 Results

****





**6.2 Conclusion**

The proposed chatbot for restaurant services has successfully demonstrated its capability to handle food ordering and table booking tasks through a conversational interface. The integration of Rasa-based Natural Language Processing (NLP) and Dialogue Management systems allowed for a smooth and efficient interaction with the users. The user interface was designed to be user-friendly and accessible, while the backend system ensured reliable and secure data storage and management. Moreover, the implementation of this project showed the potential of AI-powered chatbots in streamlining the operations of the restaurant industry and providing a convenient and personalized experience for customers. The use of Rasa framework allowed for quick development and deployment of the chatbot, making it an attractive option for organizations looking to adopt conversational AI. In light of these findings, the project has significant implications for the future of chatbots and conversational AI in the restaurant industry. Future work can focus on further improving the chatbot's capabilities, integrating additional features, and expanding its reach to other areas of the industry.

**6.3 Future Enhancements**

This project can include further improvement of the NLU module by incorporating advanced NLP techniques, incorporating additional payment gateways, integration with additional food delivery services, and implementing more robust and efficient data storage solutions. Additionally, the user interface can be improved with better design and increased interactivity to provide a better overall experience to users. Another aspect of future work could be the addition of features like personalized recommendations, menu customization, and voice recognition for hands-free ordering.

**REFERENCE**

[1]Fan, S. C., Fought, R. L., & Gahn, P. C. (2017). Adding a feature: Can a pop-up chat box enhance virtual reference services? Medical Reference Services Quarterly, 36(3), 220–228.

[2] Hsu, P., Zhao, J., Liao, K., Liu, T., & Wang, C. (2017). AllergyBot: A chatbot technology intervention for young adults with food allergies dining out. In Proceedings of the 2017 CHI conference extended abstracts on human factors in computing systems (pp. 74–79). Denver Colorado, USA: ACM.

[3] Holotescu, C. (2016). MOOCBuddy: A chatbot for personalized learning with MOOCs. In A. Iftene, & J. Vanderdonckt (Eds.), RoCHI--International conference on human computer interaction (Vol. 91, pp. 91–94), Bucharest, Romania: Matrix Rom.

[4] Abashev, A., Grigoryev, R., Grigorian, K., & Boyko, V. (2016). Programming tools for messenger-based chatbot system organization: Implication for outpatient and translational medicines. Bio Nano Science, 7(2), 403–407.

[5] S. Hussain, O. A. Sianaki, N. Ababneh, "A Survey on Conversational Agents/Chatbots Classification and Design Techniques", 2019.

[6] B. R. Ranoliya, N.Raghuwanshi, S. Singh, "Chatbot for university related FAQs," 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, 2017, pp. 1525-1530.

[7] A. Mondal, M. Dey, D. Das, S. Nagpal, K. Garda, "Chatbot: An automated conversation system for the educational domain", IEEE, 2018.

[8] R. Sharma, M. Patel, "Review on Chatbot Design Techniques in Speech Conversation Systems", Vol.5, Issue 9, September 2018.

[9] Qiu, H., Li, M., Shu, B., & Bai, B. (2020). Enhancing hospitality experience with service robots: The mediating role of rapport building. Journal of Hospitality Marketing & Management, 29(3), 247–268.

[10] Simon, O., Neuhofer, B., & Egger, R. (2020). Human-robot interaction: Conceptualising trust in frontline teams through LEGO® Serious Play®. Tourism Management Perspectives, 35, 100692.

[11] Collins, G. R. (2020). Improving human–robot interactions in hospitality settings. International Hospitality Review, 34(1), 61–79.

[12] Kervenoael, R., Hasan, R., Schwob, A., & Goh, E. (2020). Leveraging human-robot interaction in hospitality services: Incorporating the role of perceived value, empathy, and information sharing into visitors' intentions to use social robots. Tourism Management, 78, 104042.

[13] Leung, X. Y., & Wen, H. (2020). Chatbot usage in restaurant takeout orders: A comparison study of three ordering methods. Journal of Hospitality and Tourism Management, 45, 377-386.

[14] Parmar, S., Meshram, M., Parmar, P., Patel, M., & Desai, P. (2019). Smart hotel using intelligent chatbot: A review. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 5(2), 823-829.