**Development of a Basic Mini Chatbot Using**

**Python, NLTK, SpaCy, and Tkinter**

## Title

## 

**Project Title: mini Chatbot**

**Name: Yashik Srinivas**

**Date of completion:03-02-25**

**Abstract**

This report describes the creation of a basic mini chatbot using Python and the natural language processing libraries NLTK and SpaCy, along with the graphical user interface library Tkinter. The chatbot is designed to simulate conversation with users, providing responses based on predefined rules and NLP techniques. This documentation covers the design, implementation, and evaluation of the chatbot, highlighting its functionality and potential improvements.

**Table of Contents**

**s.no page.no**

1. Introduction 03
2. Objectives 03
3. Methodology 04
4. Design and Implementation 05-07
   * 4.1 Architecture
   * 4.2 Tools and Technologies
   * 4.3 Workflow
5. Results and Discussion 08
   * 5.1 Testing
   * 5.2 Evaluation
6. Future Work 08-09
7. Conclusion 09
8. References 09
9. Appendices (if any) 09-13

**Introduction**

The rise of chatbots has transformed customer service, information retrieval, and user interaction. This report provides a detailed account of developing a basic mini chatbot using Python, NLTK, SpaCy, and Tkinter. It outlines the objectives, methodologies, design, and implementation process, as well as the results and potential future enhancements.

**Objectives**

The primary objectives of this project are:

* To design and develop a basic mini chatbot using Python.
* To utilize NLTK and SpaCy for natural language processing tasks.
* To create a graphical user interface using Tkinter.
* To evaluate the chatbot's performance and identify areas for improvement.

**Methodology**

The development of the mini chatbot follows a structured approach, which includes:

1. **Requirement Analysis**: Determining the features and functionalities of the chatbot.
2. **Design**: Creating a blueprint of the chatbot’s architecture and interaction flow.
3. **Implementation**: Coding the chatbot using Python and integrating NLTK, SpaCy, and Tkinter.
4. **Testing**: Assessing the chatbot’s performance and making necessary adjustments.
5. **Deployment**: Making the chatbot available to users and monitoring its performance.

**Design and Implementation**

**4.1 Architecture**

The architecture of the chatbot consists of the following components:

* **User Interface (UI)**: The graphical interface created using Tkinter.
* **NLP Engine**: Processes user inputs using NLTK and SpaCy to understand the intent and extract relevant information.
* **Response Generator**: Generates responses based on predefined rules or machine learning models.

**4.2 Tools and Technologies**

The following tools and technologies were used:

* **Visual studio code** : Integrated development learning environment used to work on the projects
* **Python**: The primary programming language for the project.
* **NLTK**: Natural Language Toolkit for NLP tasks.
* **SpaCy**: Industrial-strength NLP library.
* **Tkinter**: Standard GUI library for Python.
* **timeit**: module for testing the performance

**4.3 Workflow**

The workflow for the chatbot development includes:

1. **Data Collection**: Gathering conversational static data for training and testing.
2. **Preprocessing**: Cleaning and preparing data for NLP tasks.
3. **Model Training**: Training the chatbot using NLTK and SpaCy.
4. **GUI Design**: Creating a user-friendly interface with Tkinter.
5. **Integration**: Integrating the chatbot with the user interface and backend.

**Flowchart**

Here’s a basic flowchart for your chatbot:

**Start**

**|**

**v**

**User Input (Tkinter GUI)**

**|**

**v**

**Preprocessing (NLTK and SpaCy)**

**|**

**v**

**NLP Engine**

**|**

**v**

**Intent Recognition**

**|**

**v**

**Response Generation**

**|**

**v**

**User Response (Tkinter GUI)**

**|**

**v**

**End**

**Results and Discussion**

**5.1 Testing**

The chatbot was tested using various test cases to evaluate its performance. The results indicated that the chatbot could effectively understand user queries and provide relevant responses.

import timeit

import nltk

import spacy

from nltk.chat.util import Chat, reflections

# Download NLTK resources

nltk.download('punkt' , quiet=True)

# Load spaCy model

nlp = spacy.load("en\_core\_web\_sm")

# Define pairs for NLTK Chat

pairs = [

(r"hi|hello|hey", ["Hello! How can I assist you today?"]),

(r"good morning", ["good morning have a blessful and energitic day how are you doing"]),

(r"good afternoon", ["good afternoon hows your day going on "]),

(r"good evening", ["good evening any plans for the evening or just a casual routine"]),

(r"what is your name?", ["I am a chatbot created for demonstration purposes."]),

(r"how are you?", ["I'm just a program, but I'm here to help you!"]),

(r"i hope you are doing good", ["I'm doing great, thank you! How about you?"]),

(r"i am doing good", ["nice to know you are doing good,working on something intresting?"]),

(r"quit", ["Goodbye! Have a great day!"]),

]

# Create a chatbot instance

chatbot = Chat(pairs, reflections)

# Function to process user input using spaCy and NLTK

def process\_input(user\_input):

doc = nlp(user\_input)

tokens = [token.text for token in doc]

response = chatbot.respond(user\_input.lower())

if response:

return response

return "I didn't quite understand that. Could you rephrase?"

# Define test case

def test\_case():

test\_inputs = [

"hi",

"good morning",

"good afternoon",

"good evening",

"what is your name?",

"how are you?",

"i hope you are doing good",

"quit"

]

for input\_text in test\_inputs:

process\_input(input\_text)

# Measure the time taken to execute the test case

execution\_time = timeit.timeit(test\_case, number=100)

print(f"Execution time for test case: {execution\_time:.4f} seconds")

**5.2 Evaluation**

The evaluation metrics included response accuracy, user satisfaction, and system robustness. The chatbot demonstrated satisfactory performance, indicating its potential for real-world applications.

**Future Work**

Future improvements for the chatbot include:

* Enhancing NLP capabilities to handle more complex queries.
* Integrating machine learning models for adaptive learning.
* Expanding the chatbot’s functionalities and applications.

**Conclusion**

This report outlines the development of a basic mini chatbot using Python, NLTK, SpaCy, and Tkinter. The chatbot successfully interacts with users, showcasing its potential for various applications. Further improvements can enhance its capabilities and performance.

**References**

* Sumit raj, " Building Chatbots with Python" Edition, Publisher, 2019.
* Online resources, such as tutorials and documentation for the tools and technologies used.

**Appendices**

* code snippets

**import nltk**

**import spacy**

**from tkinter import \***

**from nltk.chat.util import Chat, reflections**

**# Download NLTK resources**

**nltk.download('punkt',quiet=True)**

**# Load spaCy model**

**nlp = spacy.load("en\_core\_web\_sm")**

**# Define pairs for NLTK Chat**

**pairs = [**

**(r"hi|hello|hey", ["Hello! How can I assist you today?"]),**

**(r"good morning",["good morning have a blessful and energitic day how are you doing"]),**

**(r"good afternoon",["good afternoon hows your day going on "]),**

**(r"good evening",["good eveing any plans for the evening or just a casual routine"]),**

**(r"what is your name?", ["I am a chatbot created for demonstration purposes."]),**

**(r"how are you?", ["I'm just a program, but I'm here to help you!"]),**

**(r"i hope you are doing good", ["I'm doing great, thank you! How about you?"]),**

**(r"i am doing good",["nice to know you are doing good,working on something intresting?"]),**

**(r"quit", ["Goodbye! Have a great day!"]),**

**]**

**# Create a chatbot instance**

**chatbot = Chat(pairs, reflections)**

**# Function to process user input using spaCy and NLTK**

**def process\_input(user\_input):**

**doc = nlp(user\_input)**

**tokens = [token.text for token in doc]**

**response = chatbot.respond(user\_input.lower())**

**if response:**

**return response**

**return "I didn't quite understand that. Could you rephrase?"**

**# GUI for the Chatbot**

**def chatbot\_gui():**

**def send\_message():**

**user\_message = user\_input.get()**

**if user\_message.lower() == "quit":**

**chat\_log.insert(END, "Chatbot: Goodbye!\n")**

**root.quit()**

**return**

**# Display user message**

**chat\_log.insert(END, f"You: {user\_message}\n")**

**# Get chatbot response**

**bot\_response = process\_input(user\_message)**

**chat\_log.insert(END, f"Chatbot: {bot\_response}\n")**

**# Clear user input**

**user\_input.delete(0, END)**

**# Initialize GUI window**

**root = Tk()**

**root.title("Chatbot")**

**root.geometry("400x500")**

**# Chat log display**

**chat\_log = Text(root, bd=1, bg="lightgray", font=("Arial", 12))**

**chat\_log.pack(pady=55, padx=10, fill=BOTH, expand=True)**

**# User input field**

**user\_input = Entry(root, font=("Arial", 14))**

**user\_input.pack(pady=3, padx=5 , fill=X, side=LEFT, expand=True)**

**# Send button**

**send\_button = Button(root, text="Send", font=("Arial", 12), command=send\_message)**

**send\_button.pack(pady=10, padx=10, side=RIGHT)**

**# Run the GUI loop**

**root.mainloop()**

**# Start the chatbot GUI**

**chatbot\_gui()**