ABSTRACT

The Personal Assistant Chatbot is an AI-driven Python application designed to assist users in performing various digital tasks with ease. This chatbot serves as a virtual assistant capable of handling automation, information retrieval, entertainment, and system-related operations. It enhances user interaction by integrating voice-based commands, AI-powered query processing, and system automation, making it a highly efficient tool for daily activities.

With its speech recognition capabilities, the chatbot allows users to interact naturally using voice commands, while the text-to-speech engine provides clear responses. The integration of Gemini AI API further enhances its intelligence, enabling it to answer a wide range of general knowledge and contextual queries. Users can send emails, search Wikipedia, play YouTube videos, check the weather, perform mathematical calculations, create files, search locations on Google Maps, and automate web-based activities.

In addition to these functionalities, the chatbot includes interactive games, system monitoring tools, a dictionary for word definitions, and a To-Do list manager. It can also retrieve real-time news updates, jokes, and directions, ensuring a well-rounded user experience. With features like window and tab operations, screenshot capturing, volume control, and web automation, it acts as a personalized digital assistant capable of managing multiple tasks efficiently.

By leveraging external APIs and automation tools, the chatbot provides users with an intelligent and responsive system that improves productivity and simplifies routine activities. Its ability to integrate multiple services within a single interface ensures seamless interaction, making it a versatile solution for both personal and professional use.

CONTENTS

1. INTRODUCTION

1.1 ABOUT THE PROJECT

The project is entitled **“**Personal Assistant Chatbot**”**. This project is a Python-based AI-powered assistant designed to help users with various tasks such as automation, information retrieval, entertainment, and system operations.

The Personal Assistant Chatbot allows users to interact using voice commands and text input, making it an efficient tool for managing digital tasks. It utilizes speech recognition for voice input and text-to-speech for responses, creating a seamless user experience. The chatbot integrates the Google Gemini API, enabling it to process natural language queries and provide intelligent responses.

This chatbot is capable of performing multiple functions, including email sending, web searches, playing media, retrieving information from Wikipedia, weather updates, performing calculations, and automating system tasks. It also includes interactive features like games, dictionary lookups, a To-Do list manager, and system monitoring tools. Users can automate web-based activities, control system functions like opening applications and browser tabs, take screenshots, and navigate locations using Google Maps.

With a modular design and integration of various APIs, the Personal Assistant Chatbot ensures smooth task execution and enhances user productivity. It is built with a user-friendly GUI using Tkinter, making it easy to operate. This chatbot serves as a versatile personal assistant, simplifying daily tasks and providing a more interactive and automated digital experience.

2. PROBLEM DESCRIPTION

2.1 DESCRIPTION OF MODULES

To develop the Personal Assistant Chatbot, the system is divided into several modules, each responsible for handling a specific functionality. Below is an overview of the key modules:

- User Interface Module: This module is responsible for providing an interactive GUI using Tkinter. It allows users to interact with the chatbot through text and voice commands. The UI includes input fields, buttons, and display areas to show responses and task execution results.

- Speech Recognition and Response Module: This module enables voice interaction using the SpeechRecognition library for converting speech to text and pyttsx3 for text-to-speech output. It allows users to communicate naturally with the chatbot.

- AI Query Processing Module: This module integrates the Google Gemini API to handle and process user queries. It allows the chatbot to generate intelligent responses and provide relevant information based on natural language processing (NLP).

- Task Automation Module: This module is responsible for automating system-related tasks such as opening applications, managing files, controlling system volume, and taking screenshots. It uses libraries like os, subprocess, pyautogui, and psutil for system operations.

- Information Retrieval Module: This module allows the chatbot to fetch real-time information such as Wikipedia searches, weather updates, and web scraping for news. It uses Wikipedia API, Google Maps API, BeautifulSoup, and Requests to gather relevant data.

- Entertainment and Utility Module: This module includes features like playing YouTube videos, interactive games (Rock-Paper-Scissors), dictionary searches, and a To-Do list manager. It enhances the chatbot's usability by adding interactive and productive tools.

- Security and Error Handling Module: This module ensures that user interactions are handled safely and efficiently. It includes input validation, exception handling, and data security measures to prevent incorrect commands from causing system errors or unauthorized access.

- Math Module: This module performs a wide range of mathematical operations required by the user. It includes basic arithmetic (addition, subtraction, multiplication, division), advanced functions like logarithms, square roots, exponentiation, and factorial calculations. The module ensures accurate computation and efficient handling of mathematical expressions for reliable results in user queries.

By dividing the system into modules, the Personal Assistant Chatbot ensures a structured, scalable, and maintainable design, making it easy to enhance or integrate additional features in the future.

3. SYSTEM STUDY

3.1 EXISTING SYSTEM

A personal voice assistant is a software program that utilizes natural language processing and voice recognition technologies to understand and respond to spoken commands and queries. It allows users to interact with their devices, applications, and services using voice commands, and can perform a wide range of tasks such as making phone calls, scheduling appointments, setting reminders, and providing information. Some popular examples of virtual voice assistants include Amazon Alexa, Google Assistant, and Apple Siri. These AI- powered systems can be integrated with other devices and services to create a more seamless and convenient user experience.

DRAWBACKS

1. Performance: Voice assistants may have limitations in terms of their performance, such as the speed at which they can process and respond to user requests, or the complexity of tasks that they can handle.
2. Privacy and security: Voice assistants may not always clearly communicate their data collection and sharing practices to users, which could raise concerns about transparency and consent.
3. Customization: Voice assistants may not offer users a high degree of customization or control over their functionality, which could limit their usefulness and appeal to users.
4. Accuracy: Voice assistants may not always accurately understand or respond to user requests and queries, which can lead to frustration and a poor user experience.
5. Capabilities: Voice assistants may not support all tasks or functions that users may want to perform, and they may not be able to integrate with all devices or systems.

3.2 PROPOSED SYSTEM

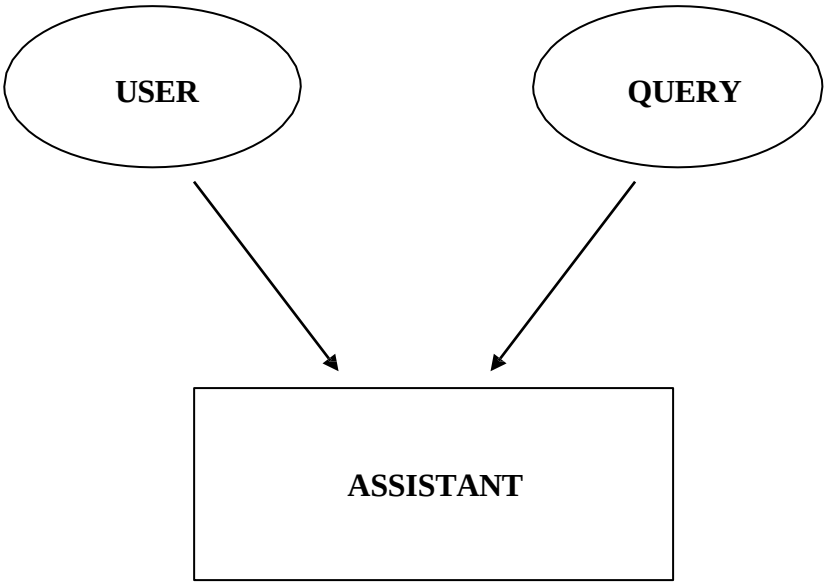
Features

- It can get some real time information such as news headlines, weather report, IP address, Internet speed, and system stats.
- It can also get entertaining contents such as jokes, latest movies or TV series, and playing songs and videos in YouTube.
- It can also generate an image from given text and can also send an email.
- It can perform system operations such as opening/closing/switching tabs, copying/pasting/deleting/selecting the text, creating a new file, taking screen shot, minimizing/maximizing/switching/closing windows.
- It can also get brief information on any topic, perform arithmetic operations, and answer any general knowledge question.
- It can perform google search, find map or distance between two places on google maps.
- We can also get the chat history along with date & time of the query.
- It can also open any installed app and some websites, we can also take notes with help of assistant.

Advantages of Proposed System

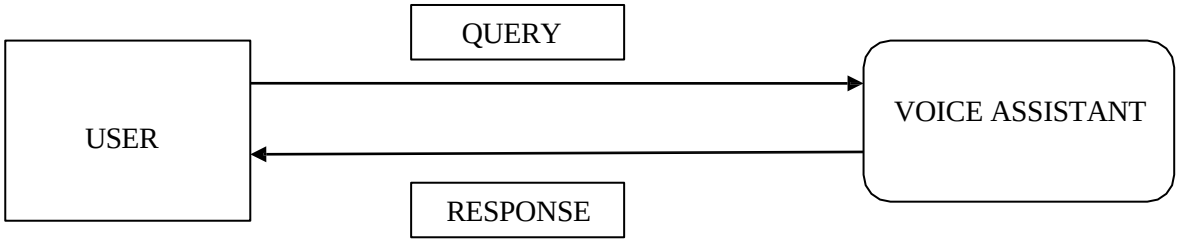
- It can create an image from the given text.
- Send email to specified mail id by dictating the message.
- It is free of cost.
- Modification can be done quickly and easily.
- It maintains privacy as user data is not shared with third parties.

3.3 ER DIAGRAM

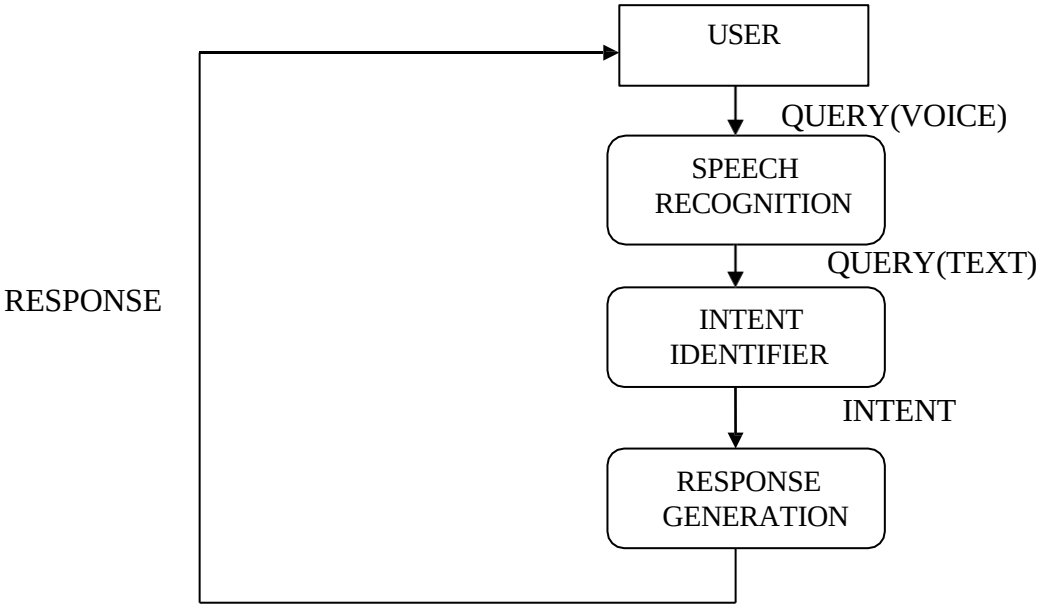


3.4 DATA FLOW DIAGRAM

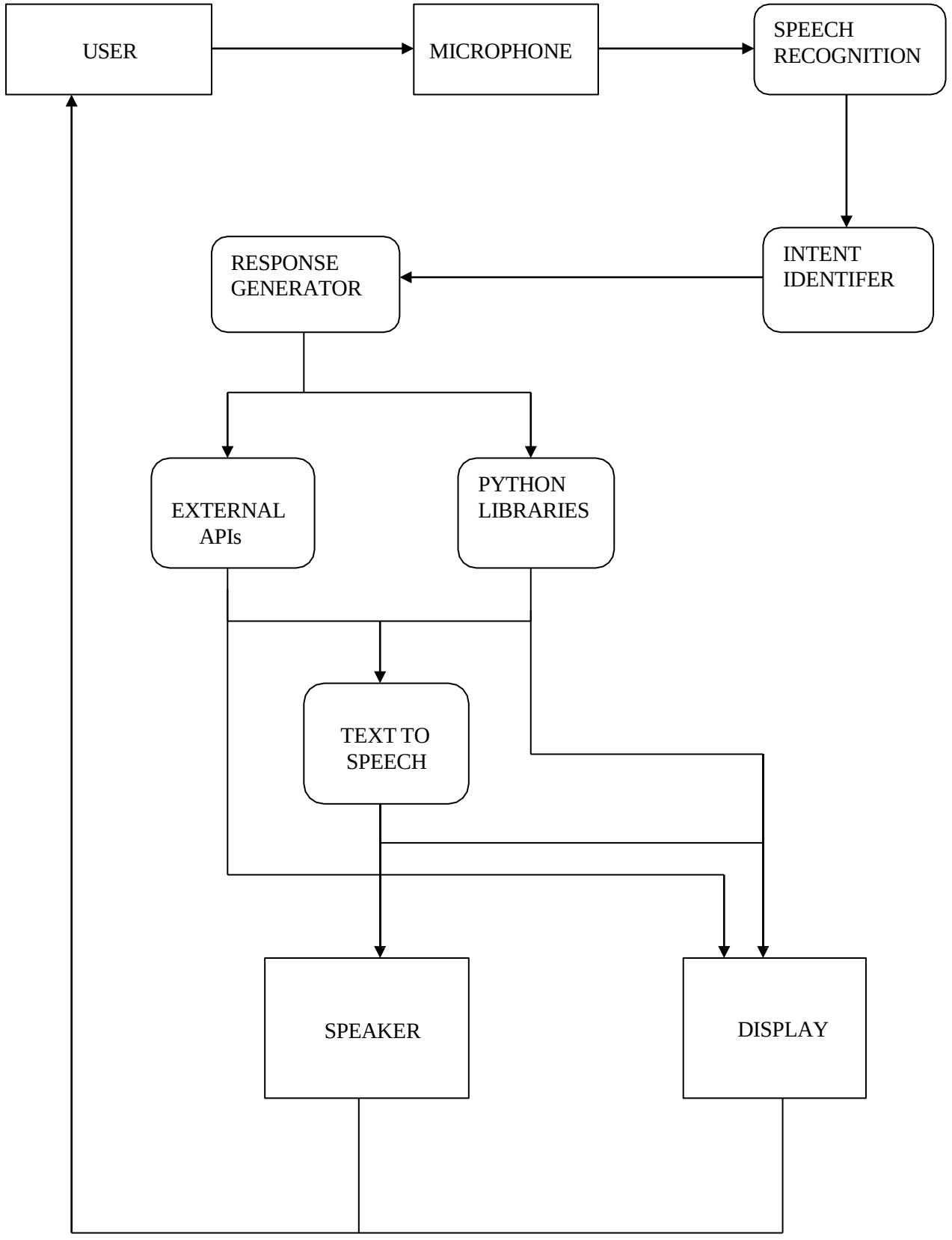
LEVEL 0



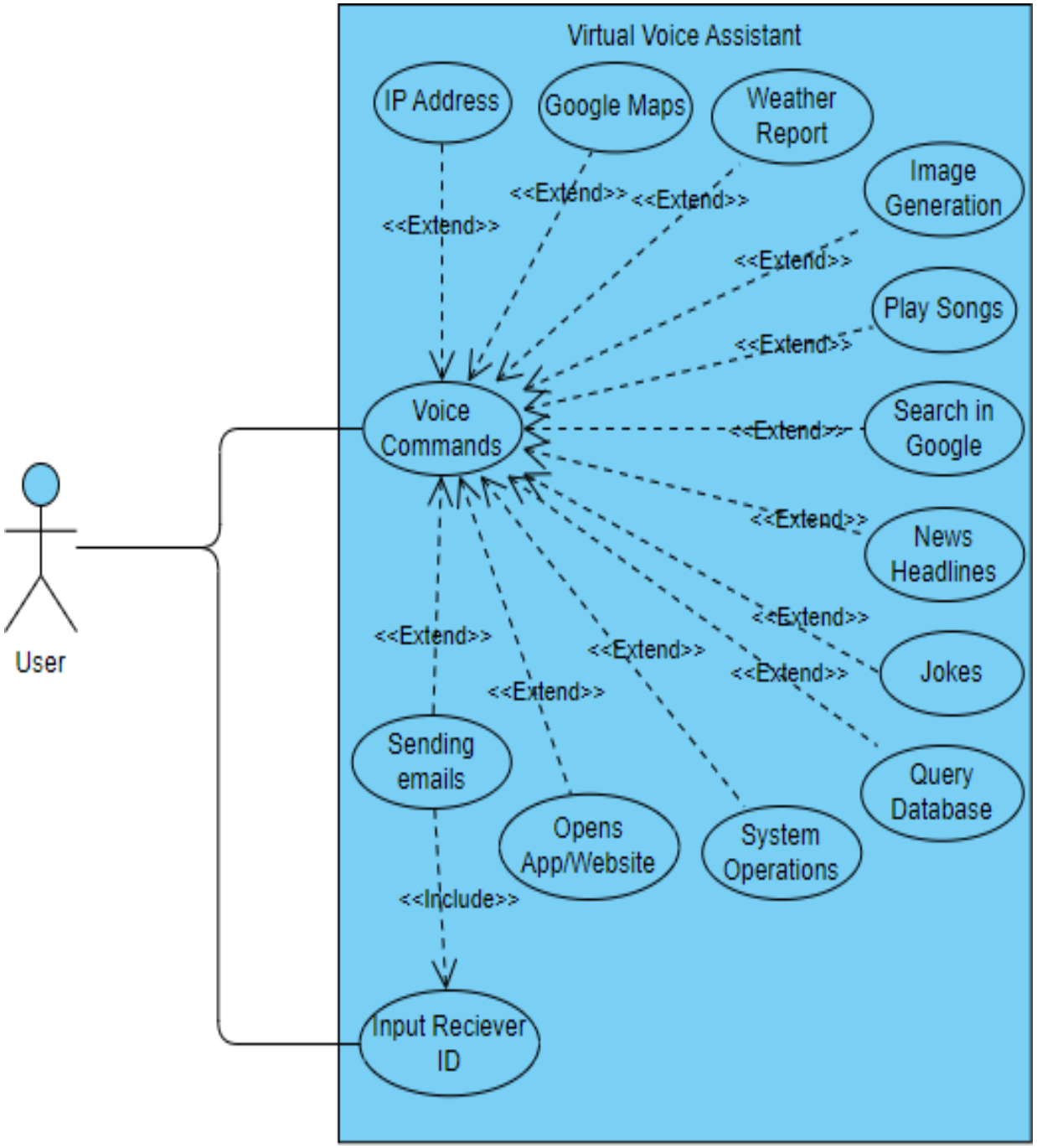
LEVEL 1



LEVEL 2


USER
MICROPHONE
SPEECH RECOGNITION
INTENT IDENTIFER
RESPONSE GENERATOR
EXTERNAL APIs
PYTHON LIBRARIES
TEXT TO SPEECH
SPEAKER
DISPLAY

3.5 USE CASE DIAGRAM



4. SYSTEM CONFIGURATION

4.1 HARDWARE SPECIFICATION

Processor	:	Pentium B950 2.10 GHz.
Hard Disk	:	250 GB
Monitor	:	14’ Colour Monitor
Mouse	:	Optical Mouse
Ram	:	2GB
Microphone	:	System In-Built (or) External

4.2 SOFTWARE SPECIFICATION

Core Language	:	PYTHON
User Interface Design	:	TKINTER
Web Browser	:	Google Chrome
Software	:	Visual Studio Code

5. OVERVIEW OF THE SOFTWARE

CORE LANGUAGE – PYTHON

Python is a high-level, interpreted programming language known for its simplicity and readability. It provides a clean syntax and supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is widely used in web development, automation, artificial intelligence, and data science due to its vast library support and active community.

Python's versatility allows it to be used in various domains, including GUI development, system scripting, and server-side programming. The chatbot is developed using Python due to its rich ecosystem of libraries and ease of integration with AI and automation tools.

FEATURES OF PYTHON

- Open-source and platform-independent
- Extensive standard library with support for automation, AI, and GUI development
- High-level and easy-to-learn syntax
- Supports multi-threading and multiprocessing for performance optimization
- Cross-platform compatibility (Windows, macOS, Linux)

USER INTERFACE DESIGN – TKINTER

Tkinter is the built-in Graphical User Interface (GUI) library in Python, used to create desktop applications with interactive features. It provides various widgets like buttons, labels, text fields, frames, and menus to build user-friendly interfaces.

FEATURES OF TKINTER

- Built into Python (no additional installation required)
- Lightweight and fast for developing GUI applications
- Provides multiple built-in widgets for UI design
- Supports event-driven programming for better user interaction
- Compatible with multiple operating systems

SOFTWARE - VISUAL STUDIO CODE

Visual Studio Code (VS Code) is a lightweight, open-source code editor developed by Microsoft. It is widely used for Python development due to its powerful extensions, debugging tools, and syntax highlighting.

FEATURES OF VISUAL STUDIO CODE

- Integrated terminal and debugger for seamless development
- Support for multiple programming languages through extensions
- Intelligent code completion with IntelliSense
- Git integration for version control
- Customizable themes and keyboard shortcuts

WEB BROWSER - GOOGLE CHROME

Google Chrome is a widely used web browser known for its speed, security, and compatibility. The chatbot uses Chrome for executing web-based automation tasks like fetching weather reports, retrieving search results, and managing voice-based interactions.

FEATURES OF GOOGLE CHROME

- Fast and secure browsing experience
- Supports JavaScript and browser automation tools (e.g., Selenium)
- Built-in developer tools for debugging web applications
- Cross-platform compatibility for seamless performance

By utilizing these technologies, the Personal Assistant Chatbot is designed to be efficient, interactive, and user-friendly, providing automation, AI-powered responses, and real-time data retrieval for an enhanced virtual assistant experience

6. DESIGN AND DEVELOPMENT

6.1 FILE DESIGN

File design is the process of structuring and organizing the chatbot's data storage for handling various functionalities such as user interactions, automation tasks, and external API responses. The objectives of file design are:

- Data Validation
- Error Handling
- User-Friendly Structure
- Efficient Storage and Retrieval

6.2 INPUT DESIGN

Input design is the process of converting user interactions into a computer-understandable format. It plays a crucial role in ensuring that the chatbot correctly interprets commands, processes data, and provides accurate responses. A poorly designed input system can lead to incorrect responses and system errors.

In the Personal Assistant Chatbot, inputs are received through voice commands using SpeechRecognition and text-based inputs via the graphical user interface (GUI). These inputs are processed and converted into structured formats before executing tasks. The system ensures:

- All voice inputs are processed accurately and converted into text.
- Text inputs are analyzed and validated before execution.
- The menu design is simple and easy to navigate, ensuring user-friendliness**.**

The chatbot's input system acts as a bridge between the user and its functionalities, ensuring that commands are processed efficiently. Well-structured input design reduces errors, enhances user interaction, and optimizes task execution.

The chatbot has been designed with efficiency in mind, minimizing manual efforts while maximizing productivity. It ensures low response time, making it a highly responsive system.

The system development was carried out considering budget constraints. Open-source technologies such as Python, Tkinter, and third-party APIs were used to ensure cost-effectiveness while maintaining high efficiency.

In addition to direct user inputs, the chatbot processes data from external sources such as Wikipedia, web search results, and weather APIs. The design of the input system ensures seamless data handling, guaranteeing that the chatbot functions optimally.

Since input errors can lead to incorrect outputs, the system has been coded with robust validation mechanisms to prevent errors and ensure smooth operation.

6.3 OUTPUT DESIGN

Output design defines the format and structure of information displayed to users. The chatbot's responses must be clear, informative, and user-friendly to ensure smooth interaction.

The Personal Assistant Chatbot provides multiple types of outputs, including:

- Text-based responses displayed on the GUI
- Voice responses generated using text-to-speech (pyttsx3)
- Automated actions such as opening applications, retrieving data, and playing media

The system's output is designed to be real-time, responsive, and well-structured. The chatbot's responses provide instant feedback, whether answering general knowledge queries, fetching news updates, or executing system commands.

Output is the key aspect of the chatbot's usability, as it determines how effectively the system communicates with users. A well-designed output format enhances the chatbot's usefulness and ensures clarity, accuracy, and easy interpretation.

Forms and console-based output are designed using Tkinter for GUI-based responses. The chatbot ensures that output formatting is visually appealing, informative, and easy to understand. The design of output responses is structured to facilitate a smooth and engaging user experience.

7. SYSTEM IMPLEMENTATION AND TESTING

SYSTEM IMPLEMENTATION

The Personal Assistant Chatbot is developed using Python, which efficiently handles data storage and retrieval. The chatbot does not require a traditional relational database like MySQL but instead utilizes local storage mechanisms and external APIs to manage data.

For handling temporary and persistent storage, the system uses:

- JSON Files – Used to store chatbot configurations, user preferences, and session data.
- Text Files – Used for saving logs and to-do list entries.
- APIs and Web Scraping – Used for retrieving real-time data, such as weather updates, Wikipedia search results, and news feeds.

Since the chatbot does not store sensitive user data, the use of lightweight storage solutions ensures fast execution and minimizes system overhead.

INTERFACE AND AUTHENTICATION USER

The Graphical User Interface (GUI) for the chatbot is designed using Tkinter, a built-in Python library that provides interactive elements such as buttons, text fields, and message displays. The GUI allows users to input queries via text or voice and receive responses in a structured manner.

- Tkinter – Used for designing an intuitive and user-friendly graphical interface.
- SpeechRecognition – Enables voice input functionality for seamless interaction.
- Pyttsx3 – Provides text-to-speech functionality for audio responses.

Since the chatbot is a standalone application, authentication is not required for general use. However, security mechanisms such as input validation and error handling are implemented to prevent misuse or unintended system failures.

SYSTEM MAINTENANCE

The maintenance phase ensures that the Personal Assistant Chatbot remains functional, efficient, and adaptable to new requirements. Proper maintenance includes:

- Regular Updates – Enhancing chatbot intelligence by updating API integrations and improving response accuracy.
- Bug Fixes – Identifying and resolving system issues to ensure smooth operation.
- Backup & Data Management – Ensuring stored data, such as user preferences and logs, is backed up to prevent loss.
- Performance Optimization – Improving processing speed and efficiency for better user experience.

System maintenance also involves enhancing the chatbot's functional capabilities, improving the user interface, and upgrading features such as speech recognition, automation, and web-based services. Ensuring proper maintenance will allow the chatbot to remain a reliable, responsive, and user-friendly virtual assistant.

.

SYSTEM TESTING

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. Testing is a process of executing a program with the intent of finding errors. The user tests the developed system and change is made according to their needs. The testing phase involves the testing of developed system using various kinds of data.

The term implementation has different meanings, ranging from the basic application, to a complete replacement of a computer system. Implementation used here is to mean the process of converting a new system design into an operational one. Conversion is one aspect of implementation.

UNIT TESTING

Unit testing was performed at the module level to ensure that individual components of the Personal Assistant Chatbot function correctly. Each module, such as speech recognition, text processing, web automation, and task execution, was tested independently to verify its accuracy and expected behavior.

FUNCTIONAL TESTING

Functional testing ensures that the Personal Assistant Chatbot performs as expected based on its defined requirements. The testing process involves verifying that each feature operates correctly under various conditions, including different inputs and real-world scenarios.

Testing Approach:

1. Identify Core Functionalities – Each feature of the chatbot, such as speech recognition, AI response generation, automation tasks, and web-based services, is tested.

2. Define Test Cases – Each function is tested using different test cases to check expected and actual outcomes.

3. Execute Test Cases – The chatbot is provided with different inputs, and its responses are observed.

4. Verify Results – The chatbot's behavior is compared to the expected results to

determine if the functionality is working correctly.

5. Log Defects and Fix Issues – If any deviations occur, the errors are logged, analyzed, and corrected before re-testing.

Functional Testing Process for Key Features:

- Speech Recognition Testing – Verifies the chatbot's ability to accurately convert speech to text.
- AI Query Processing Testing – Ensures that the chatbot correctly responds to general and contextual queries.
- Automation Testing – Checks whether the chatbot correctly executes system commands, such as opening applications, controlling volume, and setting reminders.
- Web-Based Feature Testing – Ensures that external API integrations, such as Wikipedia search, Google Maps navigation, and news retrieval, function correctly.
- Text-to-Speech Testing – Confirms that the chatbot generates proper voice responses for user queries.

Each function is tested using a variety of test cases, similar to the example provided, ensuring that the chatbot performs efficiently under different scenarios.

SYSTEM TESTING:

The A complete system test was performed to verify the overall functionality of the chatbot. The system was tested under real-world conditions, including voice commands, text queries, web searches, file operations**,** and automation tasks, ensuring that all features worked efficiently and as intended.

TEST CASE 1

Test Case No.	1
Test Type	Functional Test
Name of Test	Verify weather report feature
Test Case Description	The objective of this test case is to verify that the virtual voice assistant is able to fetch and deliver the weather report for a specified city or the current device's location.
Input	Speak "what's the weather in CITY" OR Speak "what's the weather today"
Expected Output	The virtual voice assistant should be able to deliver accurate weather reports for different cities or the current device's location.
Actual Output	The virtual voice assistant responds with news latest headlines.
Result	Pass
Comments	Working properly.

TEST CASE 2

Test Case No.	2
Test Type	Functional Test
Name of Test	Verify image fetching
Test Case Description	The objective of this test case is to verify that the virtual voice assistant is able to fetch an image based on the user's prompt.
Input	Speak "show the image" After invoking the feature a text prompt must be given "(text prompt to fetch image)"
Expected Output	"Here are the images….." "The images are …."
Actual Output	The virtual voice assistant generates image as expected.
Result	Pass
Comments	Working properly.

TEST CASE 3

Test Case No.	3
Test Type	Functional Test
Name of Test	Verify email sending feature
Test Case Description	The objective of this test case is to verify that the virtual voice assistant is able to send emails to a specified recipient.
Input	Speak “send email” Provide the recipient email address, subject, and body of the email.
Expected Output	If the email is sent successfully, the virtual voice assistant should respond with "Email has sent". If there is an error while sending the email, the virtual voice assistant should respond with "Error occurred while sending email".
Actual Output	The virtual voice assistant responds as expected.
Result	Pass
Comments	Working properly.

TEST CASE 4

Test Case No.	4
Test Type	Functional Test
Name of Test	Verify math feature
Test Case Description	The objective of this test case is to verify that the virtual voice assistant is able to solve simple mathematical equations.
Input	Speak “FACTORIAL OF 5”
Expected Output	120
Actual Output	The virtual voice assistant calculates as expected.
Result	Pass
Comments	Working properly. Works accurate for simple equation.

TEST CASE 5

Test Case No.	5
Test Type	Functional Test
Name of Test	Verify opening website and app feature
Test Case Description	The objective of this test case is to verify that the virtual voice assistant is able to open websites and apps in the user's system.
Input	Speak "open APP_NAME" OR Speak "open WEBSITE_NAME"
Expected Output	The virtual voice assistant should open websites and apps in a timely manner.
Actual Output	The virtual voice assistant opens apps and websites as expected.
Result	Pass
Comments	Working properly. Opens installed apps and websites present in websites.JSON file.

TEST CASE 6

Test Case No.	6
Test Type	Functional Test
Name of Test	Verify Google search feature
Test Case Description	The objective of this test case is to verify that the virtual voice assistant is able to perform Google searches as per user's command.
Input	Speak "search XYZ"
Expected Output	The virtual voice assistant should open a new webpage and perform a Google search for the topic specified by the user.
Actual Output	The virtual voice assistant performs Google search as expected.
Result	Pass
Comments	Working properly.

TEST CASE 7

Test Case No.	7
Test Type	Functional Test
Name of Test	Verify YouTube video play feature
Test Case Description	The objective of this test case is to verify that the virtual voice assistant is able to play specified videos on YouTube as per user's command.
Input	Speak "play XYZ video on youtube"
Expected Output	The virtual voice assistant should open a new webpage and play the specified video on YouTube.
Actual Output	The virtual voice assistant plays YouTube videos as expected.
Result	Pass
Comments	Working properly.

TEST CASE 8

Test Case No.	8
Test Type	Functional Test
Name of Test	Verify map display feature
Test Case Description	The objective of this test case is to verify that the virtual voice assistant is able to display maps of specified locations and routes as per user's command.
Input	Speak "show direction …starting location ……destination" OR Speak "show map"
Expected Output	The virtual voice assistant should open a new webpage and display the map of the specified location. OR The virtual voice assistant should open a new webpage and display the route and distance between the two specified locations.
Actual Output	The virtual voice assistant display maps and routes as expected.
Result	Pass
Comments	Working properly.

TEST CASE 9

Test Case No.	9
Test Type	Unit Test
Name of Test	Verify exit functionality
Test Case Description	The objective of this test case is to verify that the virtual voice assistant is able to terminate itself as per user's command.
Input	Speak “exit OR close”
Expected Output	The virtual voice assistant should terminate itself with exit code 0.
Actual Output	The virtual voice assistant terminate itself as expected.
Result	Pass
Comments	Working properly.

8. CONCLUSION

In conclusion, the Personal Assistant Chatbot is an AI-driven Python application designed to assist users in performing various digital tasks efficiently. The system integrates speech recognition, AI-based query processing (Gemini API), automation, and web-based services to provide a seamless virtual assistant experience.

To develop the Personal Assistant Chatbot, the proposed system follows a structured approach, including designing the chatbot architecture, implementing speech and text processing, integrating external APIs for enhanced functionality, and developing a user-friendly GUI using Tkinter.

Additionally, the chatbot is modularized into various components, including the GUI module, automation module, information retrieval module, system control module, and web-scraping module, ensuring that it is scalable, maintainable, and efficient.

By following these steps and implementing modular development, the Personal Assistant Chatbot provides users with an intelligent, interactive, and highly functional virtual assistant that improves productivity and simplifies routine digital tasks.

FUTURE ENHANCEMENT

There are several potential future enhancements that could be made to the Personal Assistant Chatbot developed in Python. Here are a few examples:

- Enhanced AI Capabilities: The chatbot could be improved by integrating more advanced AI models to provide even more accurate and context-aware responses, improving natural language understanding.
- Integration with Smart Home Devices: Future enhancements could include integration with IoT and smart home devices, allowing users to control appliances like lights, fans, and security systems using voice commands.
- Multi-language Support: The chatbot could be enhanced to support multiple languages by integrating real-time translation APIs, allowing users to communicate in their preferred language.
- Improved GUI and Mobile Application: A more interactive and feature-rich GUI could be developed using advanced frameworks like PyQt or React, and a mobile version of the chatbot could be introduced for Android and iOS platforms.
- Advanced Task Automation: The chatbot could be enhanced to schedule and manage daily tasks with calendar synchronization, appointment reminders, and smart notifications, making it a more useful personal assistant.
- Real-time Data Analysis and Insights: Future improvements could include machine learning-based user behavior analysis, providing insights and personalized recommendations based on the user's preferences and past interactions.

These enhancements would significantly improve the functionality and usability of the Personal Assistant Chatbot, making it a more intelligent, interactive, and user-friendly virtual assistant.

9. BIBILOGRAPHY

REFERENCE WEBPAGES

[1] Anonymous. Sending Emails via Gmail with Python. Script Reference, Sep 23, 2020. (https://scriptreference.com/sending-emails-via-gmail-with-python/)

[2] Amila Viraj. How To Build Your Own Chat Bot Using Deep Learning. Towards Data Science, Nov 1, 2020. (https://towardsdatascience.com/how-to-build-your-own-chatbot- using-deep-learning-bb41f970e281)

WEBSITES

- www.w3schools.com/python
- www.tutorialspoint.com

10. ANNEXURE

A. SOURCE CODE (PYTHON):

gui_assistant.py:

```
#########################
# GLOBAL VARIABLES USED #
#########################
ai_name = 'F.R.I.D.Y.'.lower()
EXIT_COMMANDS = ['bye','exit','quit','shut down', 'shutdown']

ownerName = "Cool Boy"
ownerDesignation = "Sir"
ownerPhoto = "1"
rec_email, rec_phoneno = "", ""
WAEMEntry = None

avatarChoosen = 0
choosedAvtrImage = None

botChatTextBg = "#007cc7"
botChatText = "white"
userChatTextBg = "#4da8da"

chatBgColor = '#12232e'
background = '#203647'
textColor = 'white'
AITaskStatusLblBG = '#203647'
KCS_IMG = 1 #0 for light, 1 for darkho
voice_id = 0 #0 for female, 1 for male
ass_volume = 1 #max volume
ass_voiceRate = 200 #normal voice rate

######################################## IMPORTING MODULES
########################################
""" User Created Modules """
try:
    import normalChat
    import math_function
    import appControl
    import webScrapping
    import game
    from userHandler import UserData
    from dotenv import load_dotenv
    load_dotenv()
```

```
    import timer
    import dictionary
    import ToDo
    import fileHandler
except Exception as e:
    raise e

""" System Modules """
try:
    import os
    import speech_recognition as sr
    import pyttsx3
    import threading
    import re
    import google.generativeai as genai
    from tkinter import *
    from tkinter import ttk
    from tkinter import messagebox
    from tkinter import colorchooser,Label,END
    from PIL import Image, ImageTk
    from time import sleep
    from threading import Thread
except Exception as e:
    print(e)
if os.path.exists('userData')==False:
    os.mkdir('userData')
########################################## BOOT UP WINDOW
##########################################
def ChangeSettings(write=False):
    import pickle
    global background, textColor, chatBgColor, voice_id, ass_volume, ass_voiceRate,
AITaskStatusLblBG, KCS_IMG, botChatTextBg, botChatText, userChatTextBg
    setting = {'background': background,
                                'textColor': textColor,
                                'chatBgColor': chatBgColor,
                                'AITaskStatusLblBG': AITaskStatusLblBG,
                                'KCS_IMG': KCS_IMG,
                                'botChatText': botChatText,
                                'botChatTextBg': botChatTextBg,
                                'userChatTextBg': userChatTextBg,
                                'voice_id': voice_id,
                                'ass_volume': ass_volume,
                                'ass_voiceRate': ass_voiceRate
                        }
    if write:
            with open('userData/settings.pck', 'wb') as file:
                    pickle.dump(setting, file)
            return
    try:
            with open('userData/settings.pck', 'rb') as file:
                    loadSettings = pickle.load(file)
```

```
                background = loadSettings['background']
                textColor = loadSettings['textColor']
                chatBgColor = loadSettings['chatBgColor']
                AITaskStatusLblBG = loadSettings['AITaskStatusLblBG']
                KCS_IMG = loadSettings['KCS_IMG']
                botChatText = loadSettings['botChatText']
                botChatTextBg = loadSettings['botChatTextBg']
                userChatTextBg = loadSettings['userChatTextBg']
                voice_id = loadSettings['voice_id']
                ass_volume = loadSettings['ass_volume']
                ass_voiceRate = loadSettings['ass_voiceRate']
    except Exception as e:
            pass
if os.path.exists('userData/settings.pck')==False:
    ChangeSettings(True)
def changeTheme():
    global background, textColor, AITaskStatusLblBG, KCS_IMG, botChatText,
botChatTextBg, userChatTextBg, chatBgColor
    if themeValue.get()==1:
            background, textColor, AITaskStatusLblBG, KCS_IMG = "#203647", "white",
"#203647",1
            cbl['image'] = cblDarkImg
            kbBtn['image'] = kbphDark
            settingBtn['image'] = sphDark
            AITaskStatusLbl['bg'] = AITaskStatusLblBG
            botChatText, botChatTextBg, userChatTextBg = "white", "#007cc7", "#4da8da"
            chatBgColor = "#12232e"
            colorbar['bg'] = chatBgColor
    else:
            background, textColor, AITaskStatusLblBG, KCS_IMG = "#F6FAFB", "#303E54",
"#14A769", 0
            cbl['image'] = cblLightImg
            kbBtn['image'] = kbphLight
            settingBtn['image'] = sphLight
            AITaskStatusLbl['bg'] = AITaskStatusLblBG
            botChatText, botChatTextBg, userChatTextBg = "#494949", "#EAEAEA",
"#23AE79"
            chatBgColor = "#F6FAFB"
            colorbar['bg'] = '#E8EBEF'
    root['bg'], root2['bg'] = background, background
    settingsFrame['bg'] = background
    settingsLbl['fg'], userPhoto['fg'], userName['fg'], assLbl['fg'], voiceRateLbl['fg'],
volumeLbl['fg'], themeLbl['fg'], chooseChatLbl['fg'] = textColor, textColor, textColor,
textColor, textColor, textColor, textColor, textColor
    settingsLbl['bg'], userPhoto['bg'], userName['bg'], assLbl['bg'], voiceRateLbl['bg'],
volumeLbl['bg'], themeLbl['bg'], chooseChatLbl['bg'] = background, background, background,
background, background, background, background, background
    s.configure('Wild.TRadiobutton', background=background, foreground=textColor)
    volumeBar['bg'], volumeBar['fg'], volumeBar['highlightbackground'] = background,
textColor, background
    chat_frame['bg'], root1['bg'] = chatBgColor, chatBgColor
```

```
    userPhoto['activebackground'] = background
    ChangeSettings(True)
def changeVoice(e):
    global voice_id
    voice_id=0
    if assVoiceOption.get()=='Male': voice_id=1
    engine.setProperty('voice', voices[voice_id].id)
    ChangeSettings(True)
def changeVolume(e):
    global ass_volume
    ass_volume = volumeBar.get() / 100
    engine.setProperty('volume', ass_volume)
    ChangeSettings(True)
def changeVoiceRate(e):
    global ass_voiceRate
    temp = voiceOption.get()
    if temp=='Very Low': ass_voiceRate = 100
    elif temp=='Low': ass_voiceRate = 150
    elif temp=='Fast': ass_voiceRate = 250
    elif temp=='Very Fast': ass_voiceRate = 300
    else: ass_voiceRate = 200
    print(ass_voiceRate)
    engine.setProperty('rate', ass_voiceRate)
    ChangeSettings(True)
ChangeSettings()
############################################## SET UP VOICE
##############################################
try:
    engine = pyttsx3.init()
    voices = engine.getProperty('voices')
    engine.setProperty('voice', voices[voice_id].id) #male
    engine.setProperty('volume', ass_volume)
except Exception as e:
    print(e)
######################################### SET UP TEXT TO SPEECH
#########################################
def speak(text, display=False, icon=False):
    AITaskStatusLbl['text'] = 'Speaking...'
    if icon: Label(chat_frame, image=botIcon, bg=chatBgColor).pack(anchor='w',pady=0)
    if display: attachTOframe(text, True)
    print('\n'+ai_name.upper()+': '+text)
    try:
            engine.say(text)
            engine.runAndWait()
    except:
            print("Try not to type more...")

######################################### SET UP SPEECH TO TEXT
#########################################
def record(clearChat=True, iconDisplay=True):
    print('\nListening...')
```

```
    AITaskStatusLbl['text'] = 'Listening...'
    r = sr.Recognizer()
    r.dynamic_energy_threshold = False
    r.energy_threshold = 4000
    with sr.Microphone() as source:
            r.adjust_for_ambient_noise(source)
            audio = r.listen(source)
            said = ""
            try:
                    AITaskStatusLbl['text'] = 'Processing...'
                    said = r.recognize_google(audio)
                    print(f"\nUser said: {said}")
                    if clearChat:
                            clearChatScreen()
                    if iconDisplay: Label(chat_frame, image=userIcon,
bg=chatBgColor).pack(anchor='e',pady=0)
                    attachTOframe(said)
            except Exception as e:
                    print(e)
                    # speak("I didn't get it, Say that again please...")
                    if "connection failed" in str(e):
                            speak("Your System is Offline...", True, True)
                    return 'None'
    return said.lower()
def voiceMedium():
    while True:
            query = record()
            if query == 'None': continue
            if isContain(query, EXIT_COMMANDS):
                    speak("Shutting down the System. Good Bye "+ownerDesignation+"!", True,
True)
                    break
            else: main(query.lower())
    appControl.Win_Opt('close')
def keyboardInput(e):
    user_input = UserField.get().lower()
    if user_input!="":
            clearChatScreen()
            if isContain(user_input, EXIT_COMMANDS):
                    speak("Shutting down the System. Good Bye "+ownerDesignation+"!", True,
True)
            else:
                    Label(chat_frame, image=userIcon,
bg=chatBgColor).pack(anchor='e',pady=0)
                    attachTOframe(user_input.capitalize())
                    Thread(target=main, args=(user_input,)).start()
            UserField.delete(0, END)
##################################### TASK/COMMAND HANDLER
#####################################
def isContain(txt, lst):
    for word in lst:
```

```
            if word in txt:
                return True
    return False
GEMINI_API_KEY = "YOUR GEMINI API KEY HERE"
genai.configure(api_key=GEMINI_API_KEY)
def get_gemini_response(prompt):
    """Fetch and format response from Gemini AI, ensuring it ends at a full stop when words are
between 40-50."""
    try:
        model = genai.GenerativeModel("gemini-1.5-flash")
        response = model.generate_content(prompt)

        if hasattr(response, "text"):
            full_response = response.text.strip()
            words = full_response.split()

            if len(words) <= 40:
                return full_response  # If response is already ≤40 words, return as is

            # Find the nearest full stop after 40 words, but within 50 words
            for i in range(40, min(len(words), 50)):
                if words[i].endswith('.'):
                    return " ".join(words[:i+1])  # Return up to this full stop

            # If no full stop is found, return the first 50 words as a fallback
            return " ".join(words[:50])

        return "I'm sorry, I couldn't process that request."
    except Exception as e:
        print(f"Gemini API Error: {e}")
        return "I'm sorry, I couldn't process that request."
def main(text):
            if "project" in text:
                if isContain(text, ['make', 'create']):
                    speak("What do you want to give the project name ?", True, True)
                    projectName = record(False, False)
                    speak(fileHandler.CreateHTMLProject(projectName.capitalize()),
True)
                    return

            if "create" in text and "file" in text:
                speak(fileHandler.createFile(text), True, True)
                return
            if 'list' in text:
                if isContain(text, ['add', 'create', 'make']):
                    speak("What do you want to add?", True, True)
                    item = record(False, False)
                    ToDo.toDoList(item)
                    speak("Alright, I added to your list", True)
                    return
                if isContain(text, ['show', 'my list']):
```

```
                        items = ToDo.showtoDoList()
                        if len(items)==1:
                                speak(items[0], True, True)
                                return
                        attachTOframe('\n'.join(items), True)
                        speak(items[0])
                        return
        if isContain(text, ['battery', 'system info']):
                result = appControl.OSHandler(text)
                if len(result)==2:
                        speak(result[0], True, True)
                        attachTOframe(result[1], True)
                else:
                        speak(result, True, True)
                return
        if isContain(text, ['meaning', 'dictionary', 'definition', 'define']):
                result = dictionary.translate(text)
                speak(result[0], True, True)
                if result[1]=='': return
                speak(result[1], True)
                return
        if 'volume' in text:
                appControl.volumeControl(text)
                Label(chat_frame, image=botIcon,
bg=chatBgColor).pack(anchor='w',pady=0)
                attachTOframe('Volume Settings Changed', True)
                return
        if isContain(text, ['timer', 'countdown']):
                Thread(target=timer.startTimer, args=(text,)).start()
                speak('Ok, Timer Started!', True, True)
                return
        if 'whatsapp' in text:
                speak("Sure "+ownerDesignation+"...", True, True)
                speak('Whom do you want to send the message?', True)
                WAEMPOPUP("WhatsApp", "Phone Number")
                attachTOframe(rec_phoneno)
                speak('What is the message?', True)
                message = record(False, False)
                Thread(target=webScrapping.sendWhatsapp, args=(rec_phoneno,
message,)).start()
                speak("Message is on the way. Do not move away from the screen.")
                attachTOframe("Message Sent", True)
                return
        if 'email' in text:
                speak('Whom do you want to send the email?', True, True)
                WAEMPOPUP("Email", "E-mail Address")
                attachTOframe(rec_email)
                speak('What is the Subject?', True)
                subject = record(False, False)
                speak('What message you want to send ?', True)
                message = record(False, False)
```

```
                Thread(target=webScrapping.email,  args=(rec_email,message,subject,)
).start()
                speak('Email has been Sent', True)
                return
        if isContain(text, ['covid','virus']):
                result = webScrapping.covid(text)
                if 'str' in str(type(result)):
                        speak(result, True, True)
                        return
                speak(result[0], True, True)
                result = '\n'.join(result[1])
                attachTOframe(result, True)
                return
        if isContain(text, ['youtube','video']):
                speak("Ok "+ownerDesignation+", here a video for you...", True, True)
                try:
                        speak(webScrapping.youtube(text), True)
                except Exception as e:
                        speak("Desired Result Not Found", True)
                return
        if isContain(text, ['search', 'image']):
                if 'image' in text and 'show' in text:
                        Thread(target=showImages, args=(text,)).start()
                        speak('Here are the images...', True, True)
                        return
                speak(webScrapping.googleSearch(text), True, True)
                return
        if isContain(text, ['map', 'direction']):
                if "direction" in text:
                        speak('What is your starting location?', True, True)
                        startingPoint = record(False, False)
                        speak("Ok "+ownerDesignation+", Where you want to go?", True)
                        destinationPoint = record(False, False)
                        speak("Ok "+ownerDesignation+", Getting Directions...", True)
                        try:
                                distance = webScrapping.giveDirections(startingPoint,
destinationPoint)
                                speak('You have to cover a distance of '+ distance, True)
                        except:
                                speak("I think location is not proper, Try Again!")
                else:
                        webScrapping.maps(text)
                        speak('Here you go...', True, True)
                return

        if isContain(text, ['factorial','log','value of','math',' + ',' - ',' x ','multiply','divided
by','binary','hexadecimal','octal','shift','sin ','cos ','tan ']):
                try:
                        speak(('Result is: ' + math_function.perform(text)), True, True)
                except Exception as e:
                        return
```

```
        return
    if "joke" in text:
        speak('Here is a joke...', True, True)
        speak(webScrapping.jokes(),  True)
        return
    if isContain(text, ['news']):
        speak('Getting the latest news...', True, True)
        headlines,headlineLinks = webScrapping.latestNews(2)
        for head in headlines: speak(head, True)
        speak('Do you want to read the full news?', True)
        text = record(False, False)
        if isContain(text, ["no","don't"]):
            speak("No Problem "+ownerDesignation, True)
        else:
            speak("Ok "+ownerDesignation+", Opening browser...", True)
            webScrapping.openWebsite('https://indianexpress.com/latest-news/')
            speak("You can now read the full news from this website.")
        return
    if isContain(text, ['weather']):
        data = webScrapping.weather()
        speak(", False, True)
        showSingleImage("weather", data[:-1])
        speak(data[-1])
        return
    if isContain(text, ['screenshot']):
        Thread(target=appControl.Win_Opt, args=('screenshot',)).start()
        speak("Screen Shot Taken", True, True)
        return
    if isContain(text, ['window','close that']):
        appControl.Win_Opt(text)
        return
    if isContain(text, ['tab']):
        appControl.Tab_Opt(text)
        return
    if isContain(text, ['setting']):
        raise_frame(root2)
        clearChatScreen()
        return
    if isContain(text, ['open','type','save','delete','select','press enter']):
        appControl.System_Opt(text)
        return
    if isContain(text, ['wiki', 'who is']):
        Thread(target=webScrapping.downloadImage, args=(text, 1,)).start()
        speak('Searching...', True, True)
        result = webScrapping.wikiResult(text)
        showSingleImage('wiki')
        speak(result, True)
        return
    if isContain(text, ['game']):
        speak("Which game do you want to play?", True, True)
        attachTOframe(game.showGames(), True)
```

```
                text = record(False)
                if text=="None":
                        speak("Didn't understand what you say?", True, True)
                        return
                if 'online' in text:
                        speak("Ok "+ownerDesignation+", Let's play some online games",
True, True)
                        webScrapping.openWebsite('https://www.agame.com/games/mini-
games/')
                        return
                if isContain(text, ["don't", "no", "cancel", "back", "never"]):
                        speak("No Problem "+ownerDesignation+", We'll play next time.",
True, True)
                else:
                        speak("Ok "+ownerDesignation+", Let's Play " + text, True, True)
                        os.system(f"python -c \"import game; game.play('{text}')\"")
                return
        if isContain(text, ['coin','dice','die']):
                if "toss" in text or "roll" in text or "flip" in text:
                        speak("Ok "+ownerDesignation, True, True)
                        result = game.play(text)
                        if "Head" in result: showSingleImage('head')
                        elif "Tail" in result: showSingleImage('tail')
                        else: showSingleImage(result[-1])
                        speak(result)
                        return
        if isContain(text, ['time','date']):
                speak(normalChat.chat(text), True, True)
                return
        if 'my name' in text:
                speak('Your name is, ' + ownerName, True, True)
                return
        if isContain(text, ['voice']):
                global voice_id
                try:
                        if 'female' in text: voice_id = 0
                        elif 'male' in text: voice_id = 1
                        else:
                                if voice_id==0: voice_id=1
                                else: voice_id=0
                        engine.setProperty('voice', voices[voice_id].id)
                        ChangeSettings(True)
                        speak("Hello "+ownerDesignation+", I have changed my voice. How
may I help you?", True, True)
                        assVoiceOption.current(voice_id)
                except Exception as e:
                        print(e)
                return
        if isContain(text, ['morning','evening','noon']) and 'good' in text:
                speak(normalChat.chat("good"), True, True)
                return
```

```
            result = normalChat.reply(text)
            if result != "None": speak(result, True, True)
            else:
                    ai_response = get_gemini_response(text)
                    speak(ai_response, True , True)                    #speak("Here's what I
found on the web... ", True, True)
                    #webScrapping.googleSearch(text) #uncomment this if you want to show the
result on web, means if nothing found
##################################### DELETE USER ACCOUNT
#########################################
def deleteUserData():
    result = messagebox.askquestion('Alert', 'Are you sure you want to exit ?')
    if result=='no': return
    root.destroy()
                                        ####################
                                        ####### GUI #########
                                        ####################

############ ATTACHING BOT/USER CHAT ON CHAT SCREEN ###########
def attachTOframe(text,bot=False):
    if bot:
            botchat = Label(chat_frame,text=text, bg=botChatTextBg, fg=botChatText,
justify=LEFT, wraplength=250, font=('Montserrat',12, 'bold'))
            botchat.pack(anchor='w',ipadx=5,ipady=5,pady=5)
    else:
            userchat = Label(chat_frame, text=text, bg=userChatTextBg, fg='white',
justify=RIGHT, wraplength=250, font=('Montserrat',12, 'bold'))
            userchat.pack(anchor='e',ipadx=2,ipady=2,pady=5)

def clearChatScreen():
    for wid in chat_frame.winfo_children():
            wid.destroy()
### SWITCHING BETWEEN FRAMES ###
def raise_frame(frame):
    frame.tkraise()
    clearChatScreen()
################## SHOWING DOWNLOADED IMAGES ###############
img0, img1, img2, img3, img4 = None, None, None, None, None
def showSingleImage(type, data=None):
    global img0, img1, img2, img3, img4
    try:
            img0 = ImageTk.PhotoImage(Image.open('Downloads/1.jpg').resize((90,110),
Image.Resampling.LANCZOS))
    except:
            pass
    img1 = ImageTk.PhotoImage(Image.open('extrafiles/images/heads.jpg').resize((220,200),
Image.Resampling.LANCZOS))
    img2 = ImageTk.PhotoImage(Image.open('extrafiles/images/tails.jpg').resize((220,200),
Image.Resampling.LANCZOS))
    img4 = ImageTk.PhotoImage(Image.open('extrafiles/images/WeatherImage.png'))
```

```
    if type=="weather":
            weather = Frame(chat_frame)
            weather.pack(anchor='w')
            Label(weather, image=img4, bg=chatBgColor).pack()
            Label(weather, text=data[0], font=('Arial Bold', 45), fg='white',
bg='#3F48CC').place(x=65,y=45)
            Label(weather, text=data[1], font=('Montserrat', 15), fg='white',
bg='#3F48CC').place(x=78,y=110)
            Label(weather, text=data[2], font=('Montserrat', 10), fg='white',
bg='#3F48CC').place(x=78,y=140)
            Label(weather, text=data[3], font=('Arial Bold', 12), fg='white',
bg='#3F48CC').place(x=60,y=160)
    elif type=="wiki":
            Label(chat_frame, image=img0, bg='#EAEAEA').pack(anchor='w')
    elif type=="head":
            Label(chat_frame, image=img1, bg='#EAEAEA').pack(anchor='w')
    elif type=="tail":
            Label(chat_frame, image=img2, bg='#EAEAEA').pack(anchor='w')
    else:
            img3 =
ImageTk.PhotoImage(Image.open('extrafiles/images/dice/'+type+'.jpg').resize((200,200),
Image.Resampling.LANCZOS))
            Label(chat_frame, image=img3, bg='#EAEAEA').pack(anchor='w')
def showImages(query):
    global img0, img1, img2, img3, img4
    webScrapping.downloadImage(query)
    w, h = 150, 110
    #Showing Images
    imageContainer = Frame(chat_frame, bg='#EAEAEA')
    imageContainer.pack(anchor='w')
    #loading images
    img1 = ImageTk.PhotoImage(Image.open('Downloads/1.jpg').resize((w,h),
Image.Resampling.LANCZOS))
    img2 = ImageTk.PhotoImage(Image.open('Downloads/2.jpg').resize((w,h),
Image.Resampling.LANCZOS))
    img3 = ImageTk.PhotoImage(Image.open('Downloads/3.jpg').resize((w,h),
Image.Resampling.LANCZOS))
    img4 = ImageTk.PhotoImage(Image.open('Downloads/4.jpg').resize((w,h),
Image.Resampling.LANCZOS))
    #Displaying
    Label(imageContainer, image=img1, bg='#EAEAEA').grid(row=0, column=0)
    Label(imageContainer, image=img2, bg='#EAEAEA').grid(row=0, column=1)
    Label(imageContainer, image=img3, bg='#EAEAEA').grid(row=1, column=0)
    Label(imageContainer, image=img4, bg='#EAEAEA').grid(row=1, column=1)
############################## WAEM - WhatsApp Email
###################################
def sendWAEM():
    global rec_phoneno, rec_email
    data = WAEMEntry.get()
    rec_email, rec_phoneno = data, data
    WAEMEntry.delete(0, END)
```

```
    appControl.Win_Opt('close')
def send(e):
    sendWAEM()

def WAEMPOPUP(Service='None', rec='Reciever'):
    global WAEMEntry
    PopUProot = Tk()
    PopUProot.title(f'{Service} Service')
    PopUProot.configure(bg='white')

    if Service=="WhatsApp": PopUProot.iconbitmap("extrafiles/images/whatsapp.ico")
    else: PopUProot.iconbitmap("extrafiles/images/email.ico")
    w_width, w_height = 410, 200
    s_width, s_height = PopUProot.winfo_screenwidth(), PopUProot.winfo_screenheight()
    x, y = (s_width/2)-(w_width/2), (s_height/2)-(w_height/2)
    PopUProot.geometry('%dx%d+%d+%d' % (w_width,w_height,x,y-30)) #center location of
the screen
    Label(PopUProot, text=f'Reciever {rec}', font=('Arial', 16), bg='white').pack(pady=(20, 10))
    WAEMEntry = Entry(PopUProot, bd=10, relief=FLAT, font=('Arial', 12), justify='center',
bg='#DCDCDC', width=30)
    WAEMEntry.pack()
    WAEMEntry.focus()

    SendBtn = Button(PopUProot, text='Send', font=('Arial', 12), relief=FLAT, bg='#14A769',
fg='white', command=sendWAEM)
    SendBtn.pack(pady=20, ipadx=10)
    PopUProot.bind('<Return>', send)
    PopUProot.mainloop()

######################### CHANGING CHAT BACKGROUND COLOR
#########################
def getChatColor():
    global chatBgColor
    myColor = colorchooser.askcolor()
    if myColor[1] is None: return
    chatBgColor = myColor[1]
    colorbar['bg'] = chatBgColor
    chat_frame['bg'] = chatBgColor
    root1['bg'] = chatBgColor
    ChangeSettings(True)

chatMode = 1
def changeChatMode():
    global chatMode
    if chatMode==1:
            appControl.volumeControl('mute')
            VoiceModeFrame.pack_forget()
            TextModeFrame.pack(fill=BOTH)
            UserField.focus()
            chatMode=0
    else:
```

```
            appControl.volumeControl('full')
            TextModeFrame.pack_forget()
            VoiceModeFrame.pack(fill=BOTH)
            root.focus()
            chatMode=1

#####################################  MAIN GUI
#########################################################

#### SPLASH/LOADING SCREEN ####
def progressbar():
    s = ttk.Style()
    s.theme_use('clam')
    s.configure("white.Horizontal.TProgressbar", foreground='white', background='white')
    progress_bar = ttk.Progressbar(splash_root,style="white.Horizontal.TProgressbar",
orient="horizontal",mode="determinate", length=303)
    progress_bar.pack()
    splash_root.update()
    progress_bar['value'] = 0
    splash_root.update()

    while progress_bar['value'] < 100:
            progress_bar['value'] += 5
            # splash_percentage_label['text'] = str(progress_bar['value']) + ' %'
            splash_root.update()
            sleep(0.1)

def destroySplash():
    splash_root.destroy()

if __name___== '__main__':
    splash_root = Tk()
    splash_root.configure(bg='#3895d3')
    splash_root.overrideredirect(True)
    splash_label = Label(splash_root, text="Processing...",
font=('montserrat',15),bg='#3895d3',fg='white')
    splash_label.pack(pady=40)
    # splash_percentage_label = Label(splash_root, text="0 %",
font=('montserrat',15),bg='#3895d3',fg='white')
    # splash_percentage_label.pack(pady=(0,10))

    w_width, w_height = 400, 200
    s_width, s_height = splash_root.winfo_screenwidth(), splash_root.winfo_screenheight()
    x, y = (s_width/2)-(w_width/2), (s_height/2)-(w_height/2)
    splash_root.geometry('%dx%d+%d+%d' % (w_width,w_height,x,y-30))

    progressbar()
    splash_root.after(10, destroySplash)
    splash_root.mainloop()

    root = Tk()
```

```
    root.title('F.R.I.D.A.Y')
    w_width, w_height = 400, 650
    s_width, s_height = root.winfo_screenwidth(), root.winfo_screenheight()
    x, y = (s_width/2)-(w_width/2), (s_height/2)-(w_height/2)
    root.geometry('%dx%d+%d+%d' % (w_width,w_height,x,y-30)) #center location of the
screen
    root.configure(bg=background)
    # root.resizable(width=False, height=False)
    root.pack_propagate(0)

    root1 = Frame(root, bg=chatBgColor)
    root2 = Frame(root, bg=background)
    root3 = Frame(root, bg=background)

    for f in (root1, root2, root3):
            f.grid(row=0, column=0, sticky='news')

    ################################
    ########  CHAT SCREEN  #########
    ################################

    #Chat Frame
    chat_frame = Frame(root1, width=380,height=551,bg=chatBgColor)
    chat_frame.pack(padx=10)
    chat_frame.pack_propagate(0)

    bottomFrame1 = Frame(root1, bg='#dfdfdf', height=100)
    bottomFrame1.pack(fill=X, side=BOTTOM)
    VoiceModeFrame = Frame(bottomFrame1, bg='#dfdfdf')
    VoiceModeFrame.pack(fill=BOTH)
    TextModeFrame = Frame(bottomFrame1, bg='#dfdfdf')
    TextModeFrame.pack(fill=BOTH)

    # VoiceModeFrame.pack_forget()
    TextModeFrame.pack_forget()

    cblLightImg  =  PhotoImage(file='extrafiles/images/centralButton.png')
    cblDarkImg = PhotoImage(file='extrafiles/images/centralButton1.png')
    if KCS_IMG==1: cblimage=cblDarkImg
    else: cblimage=cblLightImg
    cbl = Label(VoiceModeFrame, fg='white', image=cblimage, bg='#dfdfdf')
    cbl.pack(pady=17)
    AITaskStatusLbl = Label(VoiceModeFrame, text='    Offline', fg='white',
bg=AITaskStatusLblBG, font=('montserrat', 16))
    AITaskStatusLbl.place(x=140,y=32)

    #Settings Button
    sphLight = PhotoImage(file = "extrafiles/images/setting.png")
    sphLight = sphLight.subsample(2,2)
    sphDark = PhotoImage(file = "extrafiles/images/setting1.png")
    sphDark = sphDark.subsample(2,2)
```

```
    if KCS_IMG==1: sphimage=sphDark
    else: sphimage=sphLight
    settingBtn = Button(VoiceModeFrame,image=sphimage,height=30,width=30,
bg='#dfdfdf',borderwidth=0,activebackground="#dfdfdf",command=lambda:
raise_frame(root2))
    settingBtn.place(relx=1.0, y=30,x=-20, anchor="ne")

    #Keyboard Button
    kbphLight = PhotoImage(file = "extrafiles/images/keyboard.png")
    kbphLight = kbphLight.subsample(2,2)
    kbphDark = PhotoImage(file = "extrafiles/images/keyboard1.png")
    kbphDark = kbphDark.subsample(2,2)
    if KCS_IMG==1: kbphimage=kbphDark
    else: kbphimage=kbphLight
    kbBtn = Button(VoiceModeFrame,image=kbphimage,height=30,width=30,
bg='#dfdfdf',borderwidth=0,activebackground="#dfdfdf", command=changeChatMode)
    kbBtn.place(x=25, y=30)

    #Mic
    micImg = PhotoImage(file = "extrafiles/images/mic.png")
    micImg = micImg.subsample(2,2)
    micBtn = Button(TextModeFrame,image=micImg,height=30,width=30,
bg='#dfdfdf',borderwidth=0,activebackground="#dfdfdf", command=changeChatMode)
    micBtn.place(relx=1.0, y=30,x=-20, anchor="ne")

    #Text Field
    TextFieldImg = PhotoImage(file='extrafiles/images/textField.png')
    UserFieldLBL = Label(TextModeFrame, fg='white', image=TextFieldImg, bg='#dfdfdf')
    UserFieldLBL.pack(pady=17, side=LEFT, padx=10)
    UserField = Entry(TextModeFrame, fg='white', bg='#203647', font=('Montserrat', 16), bd=6,
width=22, relief=FLAT)
    UserField.place(x=20, y=30)
    UserField.insert(0, "Ask me anything...")
    UserField.bind('<Return>', keyboardInput)

    #User and Bot Icon
    userIcon =
PhotoImage(file="extrafiles/images/avatars/ChatIcons/a"+str(ownerPhoto)+".png")
    botIcon = PhotoImage(file="extrafiles/images/assistant2.png")
    botIcon = botIcon.subsample(2,2)


    ############################
    ########  SETTINGS  #######
    ############################

    settingsLbl = Label(root2, text='Settings', font=('Arial Bold', 15), bg=background,
fg=textColor)
    settingsLbl.pack(pady=10)
    separator = ttk.Separator(root2, orient='horizontal')
    separator.pack(fill=X)
```

```
    #User Photo
    userProfileImg = Image.open("extrafiles/images/avatars/a"+str(ownerPhoto)+".png")
    userProfileImg = ImageTk.PhotoImage(userProfileImg.resize((120, 120)))
    userPhoto = Button(root2, image=userProfileImg, bg=background, bd=0, relief=FLAT,
activebackground=background)
    userPhoto.pack(pady=(20, 5))

    #Username
    userName = Label(root2, text=ownerName, font=('Arial Bold', 15), fg=textColor,
bg=background)
    userName.pack()

    #Settings Frame
    settingsFrame = Frame(root2, width=300, height=300, bg=background)
    settingsFrame.pack(pady=20)

    assLbl = Label(settingsFrame, text='Assistant Voice', font=('Arial', 13), fg=textColor,
bg=background)
    assLbl.place(x=0, y=20)
    n = StringVar()
    assVoiceOption = ttk.Combobox(settingsFrame, values=('Female', 'Male'), font=('Arial', 13),
width=13, textvariable=n)
    assVoiceOption.current(voice_id)
    assVoiceOption.place(x=150, y=20)
    assVoiceOption.bind('<<ComboboxSelected>>', changeVoice)

    voiceRateLbl = Label(settingsFrame, text='Voice Rate', font=('Arial', 13), fg=textColor,
bg=background)
    voiceRateLbl.place(x=0, y=60)
    n2 = StringVar()
    voiceOption = ttk.Combobox(settingsFrame, font=('Arial', 13), width=13, textvariable=n2)
    voiceOption['values'] = ('Very Low', 'Low', 'Normal', 'Fast', 'Very Fast')
    voiceOption.current(ass_voiceRate//50-2) #100 150 200 250 300
    voiceOption.place(x=150, y=60)
    voiceOption.bind('<<ComboboxSelected>>',  changeVoiceRate)

    volumeLbl = Label(settingsFrame, text='Volume', font=('Arial', 13), fg=textColor,
bg=background)
    volumeLbl.place(x=0, y=105)
    volumeBar = Scale(settingsFrame, bg=background, fg=textColor, sliderlength=30,
length=135, width=16, highlightbackground=background, orient='horizontal', from_=0, to=100,
command=changeVolume)
    volumeBar.set(int(ass_volume*100))
    volumeBar.place(x=150, y=85)



    themeLbl = Label(settingsFrame, text='Theme', font=('Arial', 13), fg=textColor,
bg=background)
    themeLbl.place(x=0,y=143)
    themeValue = IntVar()
```

```
    s = ttk.Style()
    s.configure('Wild.TRadiobutton', font=('Arial Bold', 10), background=background,
foreground=textColor, focuscolor=s.configure(".")["background"])
    darkBtn = ttk.Radiobutton(settingsFrame, text='Dark', value=1, variable=themeValue,
style='Wild.TRadiobutton', command=changeTheme, takefocus=False)
    darkBtn.place(x=150,y=145)
    lightBtn = ttk.Radiobutton(settingsFrame, text='Light', value=2, variable=themeValue,
style='Wild.TRadiobutton', command=changeTheme, takefocus=False)
    lightBtn.place(x=230,y=145)
    themeValue.set(1)
    if KCS_IMG==0: themeValue.set(2)


    chooseChatLbl = Label(settingsFrame, text='Chat Background', font=('Arial', 13),
fg=textColor, bg=background)
    chooseChatLbl.place(x=0,y=180)
    cimg = PhotoImage(file = "extrafiles/images/colorchooser.png")
    cimg = cimg.subsample(3,3)
    colorbar = Label(settingsFrame, bd=3, width=18, height=1, bg=chatBgColor)
    colorbar.place(x=150, y=180)
    if KCS_IMG==0: colorbar['bg'] = '#E8EBEF'
    Button(settingsFrame, image=cimg, relief=FLAT, command=getChatColor).place(x=261,
y=180)
    backBtn = Button(settingsFrame, text='  Back  ', bd=0, font=('Arial 12'), fg='white',
bg='#14A769', relief=FLAT, command=lambda:raise_frame(root1))
    clearFaceBtn = Button(settingsFrame, text='  Close the ChatBot  ', bd=0, font=('Arial 12'),
fg='white', bg='#14A769', relief=FLAT, command=deleteUserData)
    backBtn.place(x=5, y=250)
    clearFaceBtn.place(x=120, y=250)

    try:
        # pass
        Thread(target=voiceMedium).start()
    except:
        pass
    try:
        # pass
        Thread(target=webScrapping.dataUpdate).start()
    except Exception as e:
        print('System is Offline...')

    root.iconbitmap('extrafiles/images/assistant2.ico')
    raise_frame(root1)
    root.mainloop()
```

webScarpping.py :

```
import wikipedia
import webbrowser
import requests
from bs4 import BeautifulSoup
import threading
import smtplib
import urllib.request
import os
import time
from geopy.geocoders import Nominatim
from requests.exceptions import RequestException
from geopy.distance import great_circle
from urllib.parse import urljoin

chrome_path = "C:\\Program Files\\Google\\Chrome\\Application\\chrome.exe"
webbrowser.register('chrome', None, webbrowser.BackgroundBrowser(chrome_path))

class COVID:
    def __init__(self):
        self.total = 'Not Available'
        self.deaths = 'Not Available'
        self.recovered = 'Not Available'
        self.totalIndia = 'Not Available'
        self.deathsIndia = 'Not Available'
        self.recoveredIndia = 'Not Available'

    def covidUpdate(self):
        URL = 'https://www.worldometers.info/coronavirus/'
        result = requests.get(URL)
        src = result.content
        soup = BeautifulSoup(src, 'html.parser')

        temp = []
        divs = soup.find_all('div', class_='maincounter-number')
        for div in divs:
            temp.append(div.text.strip())
        self.total, self.deaths, self.recovered = temp[0], temp[1], temp[2]
```

```
    def covidUpdateIndia(self):
        URL = 'https://www.worldometers.info/coronavirus/country/india/'
        result = requests.get(URL)
        src = result.content
        soup = BeautifulSoup(src, 'html.parser')

        temp = []
        divs = soup.find_all('div', class_='maincounter-number')
        for div in divs:
            temp.append(div.text.strip())
        self.totalIndia, self.deathsIndia, self.recoveredIndia = temp[0], temp[1],
temp[2]

    def totalCases(self,india_bool):
        if india_bool: return self.totalIndia
        return self.total

    def totalDeaths(self,india_bool):
        if india_bool: return self.deathsIndia
        return self.deaths

    def totalRecovery(self,india_bool):
        if india_bool: return self.recoveredIndia
        return self.recovered

    def symptoms(self):
        symt = ['1. Fever',
                '2. Coughing',
                '3. Shortness of breath',
                '4. Trouble breathing',
                '5. Fatigue',
                '6. Chills, sometimes with shaking',
                '7. Body aches',
                '8. Headache',
                '9. Sore throat',
                '10. Loss of smell or taste',
                '11. Nausea',
                '12. Diarrhea']
        return symt
```

```
    def prevention(self):
        prevention = ['1. Clean your hands often. Use soap and water, or an alcohol-based hand rub.',
                      '2. Maintain a safe distance from anyone who is coughing or sneezing.',
                      '3. Wear a mask when physical distancing is not possible.',
                      '4. Don’t touch your eyes, nose or mouth.',
                      '5. Cover your nose and mouth with your bent elbow or a tissue when you cough or sneeze.',
                      '6. Stay home if you feel unwell.',
                      '7. If you have a fever, cough and difficulty breathing, seek medical attention.']
        return prevention

def wikiResult(query):
    query = query.replace('wikipedia','')
    query = query.replace('search','')
    if len(query.split())==0: query = "wikipedia"
    try:
        return wikipedia.summary(query, sentences=2)
    except Exception as e:
        return "Desired Result Not Found"

class WEATHER:
    def __init__(self):
        #Currently in Lucknow, its 26 with Haze
        self.tempValue = ''
        self.city = ''
        self.currCondition = ''
        self.speakResult = ''

    def updateWeather(self):
        res = requests.get("https://ipinfo.io/")
        data = res.json()
        # URL = 'https://weather.com/en-IN/weather/today/l/'+data['loc']
        URL = 'https://weather.com/en-IN/weather/today/'
        result = requests.get(URL)
        src = result.content

        soup = BeautifulSoup(src, 'html.parser')
```

```
                city = ""
                for h in soup.find_all('h1'):
                        cty = h.text
                        cty = cty.replace('Weather','')
                        self.city = cty[:cty.find(',')]
                        break

                spans = soup.find_all('span')
                for span in spans:
                        try:
                                if span['data-testid']=="TemperatureValue":
                                        self.tempValue = span.text[:-1]
                                        break
                        except Exception as e:
                                pass

                divs = soup.find_all('div', class_='CurrentConditions--phraseValue--2xXSr')
                for div in divs:
                        self.currCondition = div.text
                        break

        def weather(self):
                from datetime import datetime
                today = datetime.today().strftime('%A')
                self.speakResult = "Currently in " + self.city + ", its " + self.tempValue + "
degree, with a " + self.currCondition
                return [self.tempValue, self.currCondition, today, self.city, self.speakResult]

c = COVID()
w = WEATHER()

def dataUpdate():
        c.covidUpdate()
        c.covidUpdateIndia()
        w.updateWeather()

##### WEATHER #####
def weather():
        return w.weather()
```

```
### COVID ###
def covid(query):

    if "india" in query: india_bool = True
    else: india_bool = False

    if "statistic" in query or 'report' in query:
        return ["Here are the statistics...", ["Total cases: " +
c.totalCases(india_bool), "Total Recovery: " + c.totalRecovery(india_bool), "Total
Deaths: " + c.totalDeaths(india_bool)]]

    elif "symptom" in query:
        return ["Here are the Symptoms...", c.symptoms()]

    elif "prevent" in query or "measure" in query or "precaution" in query:
        return ["Here are the some of preventions from COVID-19:", c.prevention()]

    elif "recov" in query:
        return "Total Recovery is: " + c.totalRecovery(india_bool)

    elif "death" in query:
        return "Total Deaths are: " + c.totalDeaths(india_bool)

    else:
        return "Total Cases are: " + c.totalCases(india_bool)

def latestNews(news=5):
    URL = 'https://indianexpress.com/latest-news/'
    result = requests.get(URL)
    src = result.content

    soup = BeautifulSoup(src, 'html.parser')

    headlineLinks = []
    headlines = []

    divs = soup.find_all('div', {'class':'title'})

    count=0
```

```
        for div in divs:
                count += 1
                if count>news:
                        break
                a_tag = div.find('a')
                headlineLinks.append(a_tag.attrs['href'])
                headlines.append(a_tag.text)

        return headlines,headlineLinks

def maps(text):
        text = text.replace('maps', '')
        text = text.replace('map', '')
        text = text.replace('google', '')
        openWebsite('https://www.google.com/maps/place/'+text)

def giveDirections(startingPoint, destinationPoint):

        geolocator = Nominatim(user_agent='assistant')
        if 'current' in startingPoint:
                res = requests.get("https://ipinfo.io/")
                data = res.json()
                startinglocation = geolocator.reverse(data['loc'])
        else:
                startinglocation = geolocator.geocode(startingPoint)

        destinationlocation = geolocator.geocode(destinationPoint)
        startingPoint = startinglocation.address.replace(' ', '+')
        destinationPoint = destinationlocation.address.replace(' ', '+')

        openWebsite('https://www.google.co.in/maps/dir/'+startingPoint+'/'+destinationPoi
nt+'/')

        startinglocationCoordinate = (startinglocation.latitude, startinglocation.longitude)
        destinationlocationCoordinate = (destinationlocation.latitude,
destinationlocation.longitude)
        total_distance = great_circle(startinglocationCoordinate,
destinationlocationCoordinate).km #.mile
        return str(round(total_distance, 2)) + 'KM'

def openWebsite(url='https://www.google.com/'):
```

```
        webbrowser.get('chrome').open(url)

def jokes():
        URL = 'https://icanhazdadjoke.com/'
        result = requests.get(URL)
        src = result.content

        soup = BeautifulSoup(src, 'html.parser')

        try:
                p = soup.find('p')
                return p.text
        except Exception as e:
                raise e

def youtube(query):
    from youtube_search import YoutubeSearch

    query = query.lower().strip()  # Convert to lowercase and remove extra spaces

    # If the query is "open youtube", open the homepage
    if query in ["open youtube", "open youtube.com", "go to youtube"]:
        openWebsite("https://www.youtube.com")
        return "Opening YouTube..."

    # Clean up query for search
    query = query.replace('play', ' ')
    query = query.replace('on youtube', ' ')
    query = query.replace('youtube', ' ')

    # Perform YouTube search
    results = YoutubeSearch(query, max_results=1).to_dict()

    # Open the first search result
    if results:
        openWebsite('https://www.youtube.com/watch?v=' + results[0]['id'])
        return "Enjoy Sir..."
    else:
        return "Sorry, I couldn't find any results on YouTube."
```

```
def googleSearch(query):
        if 'image' in query:
                query += "&tbm=isch"
        query = query.replace('images','')
        query = query.replace('image','')
        query = query.replace('search','')
        query = query.replace('show','')
        openWebsite("https://www.google.com/search?q=" + query)
        return "Here you go..."

def sendWhatsapp(phone_no='',message=''):
        phone_no = '+91' + str(phone_no)
        openWebsite('https://web.whatsapp.com/send?phone='+phone_no+'&text='+messag
e)
        import time
        from pynput.keyboard import Key, Controller
        time.sleep(10)
        k = Controller()
        k.press(Key.enter)

def email(rec_email=None, text="Hello, It's F.R.I.D.A.Y. here...", sub='F.R.I.D.A.Y.'):
        USERNAME = os.getenv('MAIL_USERNAME') # email address
        PASSWORD = os.getenv('MAIL_PASSWORD')
        if not USERNAME or not PASSWORD:
                raise Exception("MAIL_USERNAME or MAIL_PASSWORD are not
loaded in environment, create a .env file and add these 2 values")

        if '@gmail.com' not in rec_email: return
        s = smtplib.SMTP('smtp.gmail.com', 587)
        s.starttls()
        s.login(USERNAME, PASSWORD)
        message = 'Subject: {}\n\n{}'.format(sub, text)
        s.sendmail(USERNAME, rec_email, message)
        print("Sent")
        s.quit()



def get_with_retry(url, headers, retries=3, delay=5):
```

```
    """Retry mechanism for making requests."""
    for i in range(retries):
        try:
            response = requests.get(url, headers=headers, timeout=10)
            response.raise_for_status()
            return response
        except RequestException as e:
            print(f"Attempt {i+1} failed: {e}")
            time.sleep(delay)
    return None  # Return None if all attempts fail

def downloadImage(query, n=5):
    """Downloads n images based on a Google Image search query."""
    query = query.replace('images', '').replace('image', '').replace('search', '').replace('show',
'')
    URL = "https://www.google.com/search?tbm=isch&q=" + query

    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36'
    }

    result = get_with_retry(URL, headers)
    if not result:
        print("Failed to fetch image search results.")
        return

    soup = BeautifulSoup(result.content, 'html.parser')
    imgTags = soup.find_all('img')

    if not os.path.exists('Downloads'):
        os.mkdir('Downloads')

    count = 0
    for i in imgTags:
        if count == n:
            break
        try:
            img_url = i.get('src')
            if not img_url:
```

```
                continue

            # Ensure the image URL is absolute
            img_url = urljoin(URL, img_url)

            # Download the image
            urllib.request.urlretrieve(img_url, f'Downloads/{count}.jpg')
            count += 1
            print(f'Downloaded {count} images')
        except Exception as e:
            print(f"Error downloading image {count}: {e}")
            continue
```

filehandler.py:

```
import subprocess
import wmi
import os
import sys
import webbrowser

if os.path.exists('Files and Document') == False:
    os.mkdir('Files and Document')
path = 'Files and Document/'

def isContain(text, list):
    for word in list:
        if word in text:
            return True
    return False

def createFile(text):
    appLocation = "C:\\Users\\jainul\\AppData\\Local\\Programs\\Microsoft VS
Code\\Code.exe"

    if isContain(text, ["ppt","power point","powerpoint"]):
        file_name = "sample_file.ppt"
        appLocation = "C:\\Program Files (x86)\\Microsoft
Office\\Office12\\POWERPNT.EXE"

    elif isContain(text, ['excel','spreadsheet']):
        file_name = "sample_file.xsl"
        appLocation = "C:\\Program Files (x86)\\Microsoft
Office\\Office12\\EXCEL.EXE"
```

```
        elif isContain(text, ['word','document']):
                file_name = "sample_file.docx"
                appLocation = "C:\\Program Files (x86)\\Microsoft
Office\\Office12\\WINWORD.EXE"

        elif isContain(text, ["text","simple","normal"]): file_name = "sample_file.txt"
        elif "python" in text: file_name = "sample_file.py"
        elif "css" in text:     file_name = "sample_file.css"
        elif "javascript" in text: file_name = "sample_file.js"
        elif "html" in text: file_name = "sample_file.html"
        elif "c plus plus" in text or "c++" in text: file_name = "sample_file.cpp"
        elif "java" in text: file_name = "sample_file.java"
        elif "json" in text: file_name = "sample_file.json"
        else: return "Unable to create this type of file"

        file = open(path + file_name, 'w')
        file.close()
        subprocess.Popen([appLocation, path + file_name])
        return "File is created.\nNow you can edit this file"

def CreateHTMLProject(project_name='Sample'):

        if os.path.isdir(path + project_name):
                webbrowser.open(os.getcwd() + '/' + path + project_name + "\\index.html")
                return 'There is a same project which is already created, look at this...'
        else:
           os.mkdir(path + project_name)

        os.mkdir(path+project_name+ '/images')
        os.mkdir(path+project_name+ '/videos')

        htmlContent = '<html>\n\t<head>\n\t\t<title> ' + project_name + '
</title>\n\t\t<link rel="stylesheet" type="text/css"
href="style.css">\n\t</head>\n<body>\n\t<p id="label"></p>\n\t<button id="btn"
onclick="showText()"> Click Me </button>\n\t<script
src="script.js"></script>\n</body>\n</html>'

        htmlFile = open(path+project_name+ '/index.html', 'w')
        htmlFile.write(htmlContent)
        htmlFile.close()

        cssContent = '* {\n\tmargin:0;\n\tpadding:0;\n}\nbody
{\n\theight:100vh;\n\tdisplay:flex;\n\tjustify-content:center;\n\talign-
items:center;\n}\n#btn {\n\twidth:200px;\n\tpadding: 20px 10px;\n\tborder-
radius:5px;\n\tbackground-color:red;\n\tcolor:#fff;\n\toutline:none;border:none;\n}\np
{\n\tfont-size:30px;\n}'

        cssFile = open(path+project_name+ '/style.css', 'w')
```

```
        cssFile.write(cssContent)
        cssFile.close

        jsContent = 'function showText()
    {\n\tdocument.getElementById("label").innerHTML="Successfully Created '+
    project_name +' Project";\n\tdocument.getElementById("btn").style="background-
    color:green;"\n}'

        jsFile = open(path+project_name+ '/script.js', 'w')
        jsFile.write(jsContent)
        jsFile.close()

        appLocation = "C:\\Users\\jainul\\AppData\\Local\\Programs\\Microsoft VS
    Code\\Code.exe"
        # subprocess.Popen([appLocation, path + project_name])
        subprocess.Popen([appLocation, path + project_name + "/index.html"])
        subprocess.Popen([appLocation, path + project_name + "/style.css"])
        subprocess.Popen([appLocation, path + project_name + "/script.js"])

        webbrowser.open(os.getcwd() + '/' + path + project_name + "\\index.html")


        return f'Successfully Created {project_name} Project'
```

appControl.py:

```
import pyscreenshot as ImageGrab
import time
import webbrowser
import playsound
import os
import subprocess
from pynput.keyboard import Key, Controller
import psutil

chrome_path = "C:\\Program Files\\Google\\Chrome\\Application\\chrome.exe"
webbrowser.register('chrome', None, webbrowser.BackgroundBrowser(chrome_path))

def openWebsite(url='https://www.google.com/'):
    webbrowser.get('chrome').open(url)

class SystemTasks:
    def __init__(self):
        self.keyboard = Controller()

    def openApp(self, appName):
        appName = appName.replace('paint', 'mspaint')
        appName = appName.replace('wordpad', 'write')
```

```
            appName = appName.replace('word', 'write')
            appName = appName.replace('calculator', 'calc')
            try: subprocess.Popen('C:\\Windows\\System32\\'+appName[5:]+'.exe')
            except: pass

    def write(self, text):
            text = text[5:]
            for char in text:
                    self.keyboard.type(char)
                    time.sleep(0.02)

    def select(self):
            self.keyboard.press(Key.ctrl)
            self.keyboard.press('a')
            self.keyboard.release('a')
            self.keyboard.release(Key.ctrl)

    def hitEnter(self):
            self.keyboard.press(Key.enter)
            self.keyboard.release(Key.enter)

    def delete(self):
            self.keyboard.press(Key.backspace)
            self.keyboard.release(Key.enter)

    def save(self, text):
            if "don't" in text:
                    self.keyboard.press(Key.right)
            else:
                    self.keyboard.press(Key.ctrl)
                    self.keyboard.press('s')
                    self.keyboard.release('s')
                    self.keyboard.release(Key.ctrl)
            self.hitEnter()

class TabOpt:
    def __init__(self):
            self.keyboard = Controller()

    def switchTab(self):
            self.keyboard.press(Key.ctrl)
            self.keyboard.press(Key.tab)
            self.keyboard.release(Key.tab)
            self.keyboard.release(Key.ctrl)

    def closeTab(self):
            self.keyboard.press(Key.ctrl)
            self.keyboard.press('w')
            self.keyboard.release('w')
```

```
            self.keyboard.release(Key.ctrl)

        def newTab(self):
            self.keyboard.press(Key.ctrl)
            self.keyboard.press('n')
            self.keyboard.release('n')
            self.keyboard.release(Key.ctrl)


class WindowOpt:
    def __init__(self):
        self.keyboard = Controller()

    def openWindow(self):
        self.maximizeWindow()

    def closeWindow(self):
        self.keyboard.press(Key.alt_l)
        self.keyboard.press(Key.f4)
        self.keyboard.release(Key.f4)
        self.keyboard.release(Key.alt_l)

    def minimizeWindow(self):
        for i in range(2):
            self.keyboard.press(Key.cmd)
            self.keyboard.press(Key.down)
            self.keyboard.release(Key.down)
            self.keyboard.release(Key.cmd)
            time.sleep(0.05)

    def maximizeWindow(self):
        self.keyboard.press(Key.cmd)
        self.keyboard.press(Key.up)
        self.keyboard.release(Key.up)
        self.keyboard.release(Key.cmd)

    def moveWindow(self, operation):
        self.keyboard.press(Key.cmd)

        if "left" in operation:
            self.keyboard.press(Key.left)
            self.keyboard.release(Key.left)
        elif "right" in operation:
            self.keyboard.press(Key.right)
            self.keyboard.release(Key.right)
        elif "down" in operation:
            self.keyboard.press(Key.down)
            self.keyboard.release(Key.down)
        elif "up" in operation:
```

```
                        self.keyboard.press(Key.up)
                        self.keyboard.release(Key.up)
                self.keyboard.release(Key.cmd)

        def switchWindow(self):
                self.keyboard.press(Key.alt_l)
                self.keyboard.press(Key.tab)
                self.keyboard.release(Key.tab)
                self.keyboard.release(Key.alt_l)


        def takeScreenShot(self):
                from random import randint
                im = ImageGrab.grab()
                im.save(f'Files and Document/ss_{randint(1, 100)}.jpg')
                playsound.playsound("extrafiles/audios/photoclick.mp3")

def isContain(text, lst):
        for word in lst:
                if word in text:
                        return True
        return False

def Win_Opt(operation):
        w = WindowOpt()
        if isContain(operation, ['open']):
                w.openWindow()
        elif isContain(operation, ['close']):
                w.closeWindow()
        elif isContain(operation, ['mini']):
                w.minimizeWindow()
        elif isContain(operation, ['maxi']):
                w.maximizeWindow()
        elif isContain(operation, ['move', 'slide']):
                w.moveWindow(operation)
        elif isContain(operation, ['switch','which']):
                w.switchWindow()
        elif isContain(operation, ['screenshot','capture','snapshot']):
                w.takeScreenShot()
        return

def Tab_Opt(operation):
        t = TabOpt()
        if isContain(operation, ['new','open','another','create']):
                t.newTab()
        elif isContain(operation, ['switch','move','another','next','previous','which']):
                t.switchTab()
        elif isContain(operation, ['close','delete']):
                t.closeTab()
```

```
        else:
                return


def System_Opt(operation):
        s = SystemTasks()
        if 'delete' in operation:
                s.delete()
        elif 'save' in operation:
                s.save(operation)
        elif 'type' in operation:
                s.write(operation)
        elif 'select' in operation:
                s.select()
        elif 'enter' in operation:
                s.hitEnter()
        elif isContain(operation, ['notepad','paint','calc','word']):
                s.openApp(operation)
        elif isContain(operation, ['music','video']):
                s.playMusic(operation)
        else:
                open_website(operation)
                return
################################
###########  VOLUME ###########
################################
keyboard = Controller()
def mute():
        for i in range(50):
                keyboard.press(Key.media_volume_down)
                keyboard.release(Key.media_volume_down)
def full():
        for i in range(50):
                keyboard.press(Key.media_volume_up)
                keyboard.release(Key.media_volume_up)
def volumeControl(text):
        if 'full' in text or 'max' in text: full()
        elif 'mute' in text or 'min' in text: mute()
        elif 'incre' in text:
                for i in range(5):
                        keyboard.press(Key.media_volume_up)
                        keyboard.release(Key.media_volume_up)
        elif 'decre' in text:
                for i in range(5):
                        keyboard.press(Key.media_volume_down)
                        keyboard.release(Key.media_volume_down)

def systemInfo():
        import wmi
```

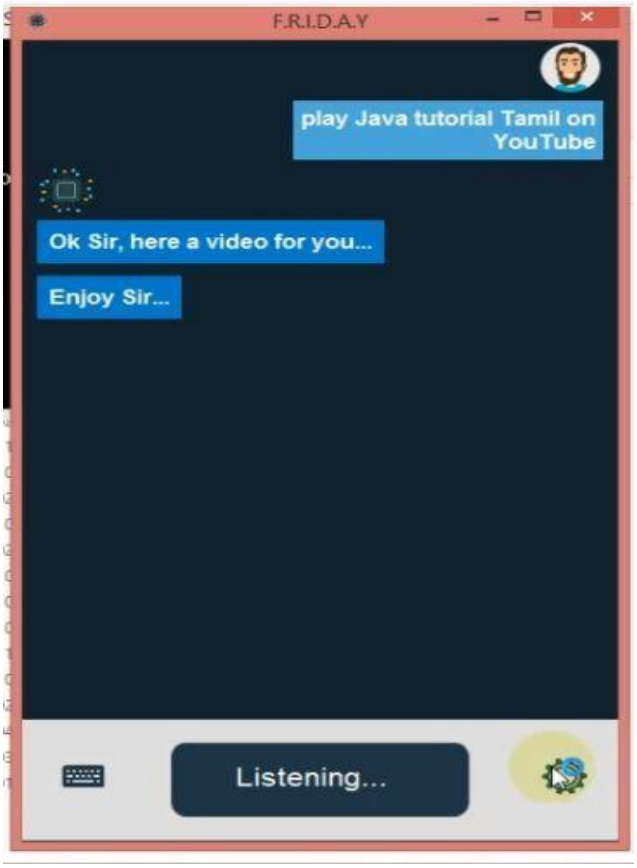
```
        c = wmi.WMI()
        my_system_1 = c.Win32_LogicalDisk()[0]
        my_system_2 = c.Win32_ComputerSystem()[0]
        info = ["Total Disk Space: " + str(round(int(my_system_1.Size)/(1024**3),2)) + "
GB",
                        "Free Disk Space: " +
str(round(int(my_system_1.Freespace)/(1024**3),2)) + " GB",
                        "Manufacturer: " + my_system_2.Manufacturer,
                        "Model: " + my_system_2. Model,
                        "Owner: " + my_system_2.PrimaryOwnerName,
                        "Number of Processors: " + str(my_system_2.NumberOfProcessors),
                        "System Type: " + my_system_2.SystemType]
        return info

def batteryInfo():
        # usage = str(psutil.cpu_percent(interval=0.1))
        battery = psutil.sensors_battery()
        pr = str(battery.percent)
        if battery.power_plugged:
                return "Your System is currently on Charging Mode and it's " + pr + "%
done."
        return "Your System is currently on " + pr + "% battery life."

def OSHandler(query):
        if isContain(query, ['system', 'info']):
                return ['Here is your System Information...', '\n'.join(systemInfo())]
        elif isContain(query, ['cpu', 'battery']):
                return batteryInfo()


from difflib import get_close_matches
import json
from random import choice


data = json.load(open('extrafiles/websites.json', encoding='utf-8'))

def open_website(query):
        query = query.replace('open','')
        if query in data:
                response = data[query]
        else:
                query = get_close_matches(query, data.keys(), n=2, cutoff=0.5)
                if len(query)==0: return "None"
                response = choice(data[query[0]])
        openWebsite(response)
```

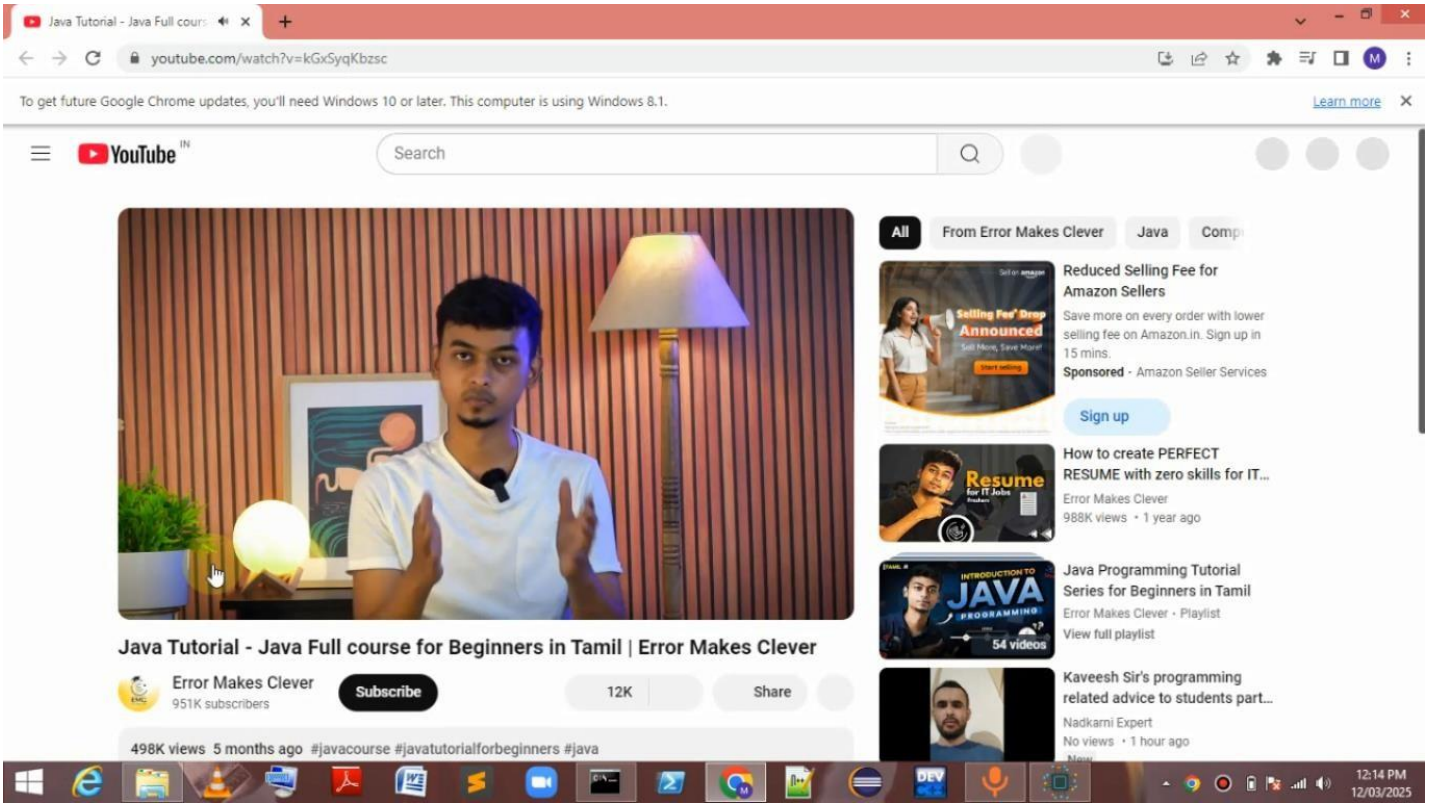
math_function.py:

```
        import math
def basicOperations(text):
        if 'root' in text:
                temp = text.rfind(' ')
                num = int(text[temp+1:])
                return round(math.sqrt(num),2)

        text = text.replace('plus', '+')
        text = text.replace('minus', '-')
        text = text.replace('x', '*')
        text = text.replace('multiplied by', '*')
        text = text.replace('multiply', '*')
        text = text.replace('divided by', '/')
        text = text.replace('to the power', '**')
        text = text.replace('power', '**')
        result = eval(text)
        return round(result,2)

def bitwiseOperations(text):
        if 'right shift' in text:
                temp = text.rfind(' ')
                num = int(text[temp+1:])
                return num>>1
        elif 'left shift' in text:
                temp = text.rfind(' ')
                num = int(text[temp+1:])
                return num<<1
        text = text.replace('and', '&')
        text = text.replace('or', '|')
        text = text.replace('not of', '~')
        text = text.replace('not', '~')
        text = text.replace('xor', '^')
        result = eval(text)
        return result

def conversions(text):
        temp = text.rfind(' ')
        num = int(text[temp+1:])
        if 'bin' in text:
                return eval('bin(num)')[2:]
        elif 'hex' in text:
                return eval('hex(num)')[2:]
        elif 'oct' in text:
                return eval('oct(num)')[2:]
```

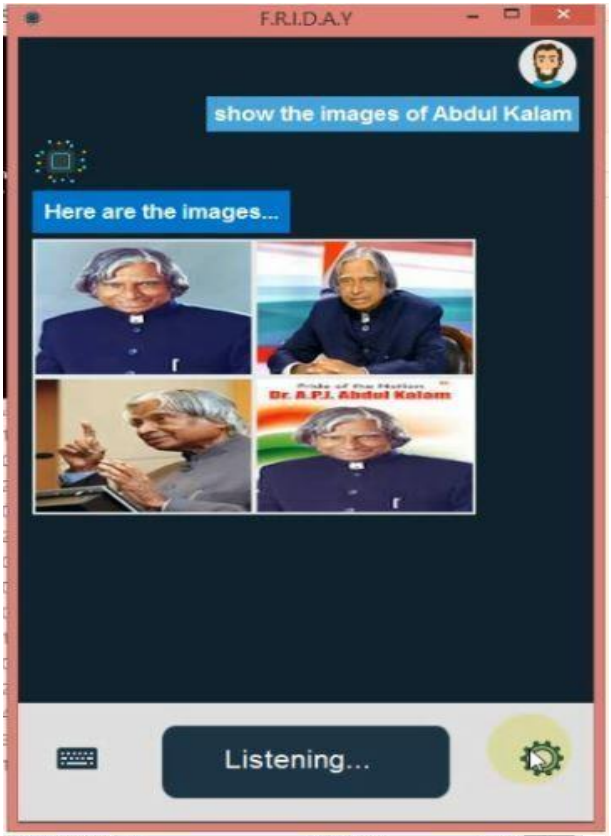
```
def trigonometry(text):
	temp = text.replace('degree','')
	temp = text.rfind(' ')
	deg = int(text[temp+1:])
	rad = (deg * math.pi) / 180
	if 'sin' in text:
		return round(math.sin(rad),2)
	elif 'cos' in text:
		return round(math.cos(rad),2)
	elif 'tan' in text:
		return round(math.tan(rad),2)

def factorial(n):
	if n==1 or n==0: return 1
	else: return n*factorial(n-1)

def logFind(text):
	temp = text.rfind(' ')
	num = int(text[temp+1:])
	return round(math.log(num,10),2)

def isHaving(text, lst):
	for word in lst:
		if word in text:
			return True
	return False

def perform(text):
	text = text.replace('math','')
	if "factorial" in text: return str(factorial(int(text[text.rfind(' ')+1:])))
	elif isHaving(text, ['sin','cos','tan']): return str(trigonometry(text))
	elif isHaving(text, ['bin','hex','oct']): return str(conversions(text))
	elif isHaving(text, ['shift','and','or','not']): return str(bitwiseOperations(text))
	elif 'log' in text: return str(logFind(text))
	else: return str(basicOperations(text))
```

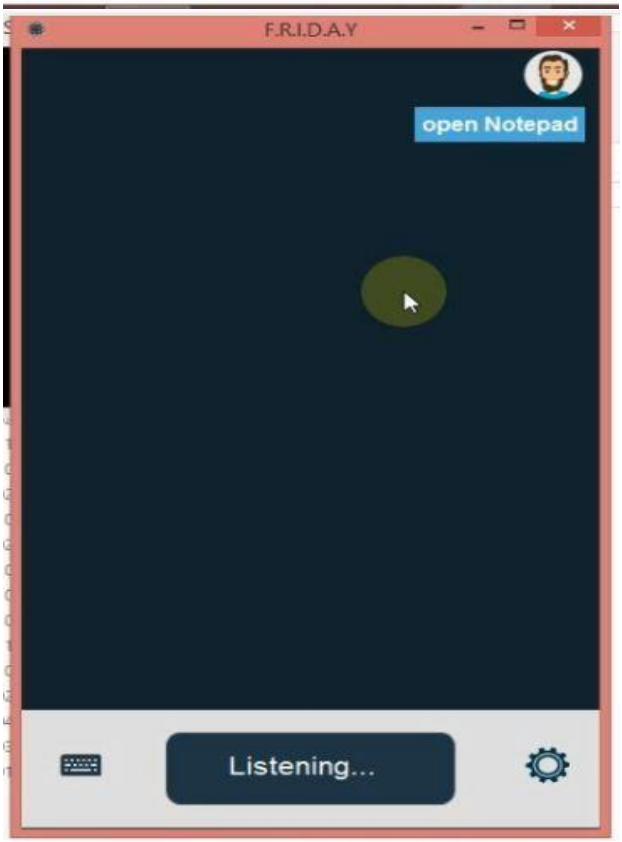
B. SCREEN INTERFACES



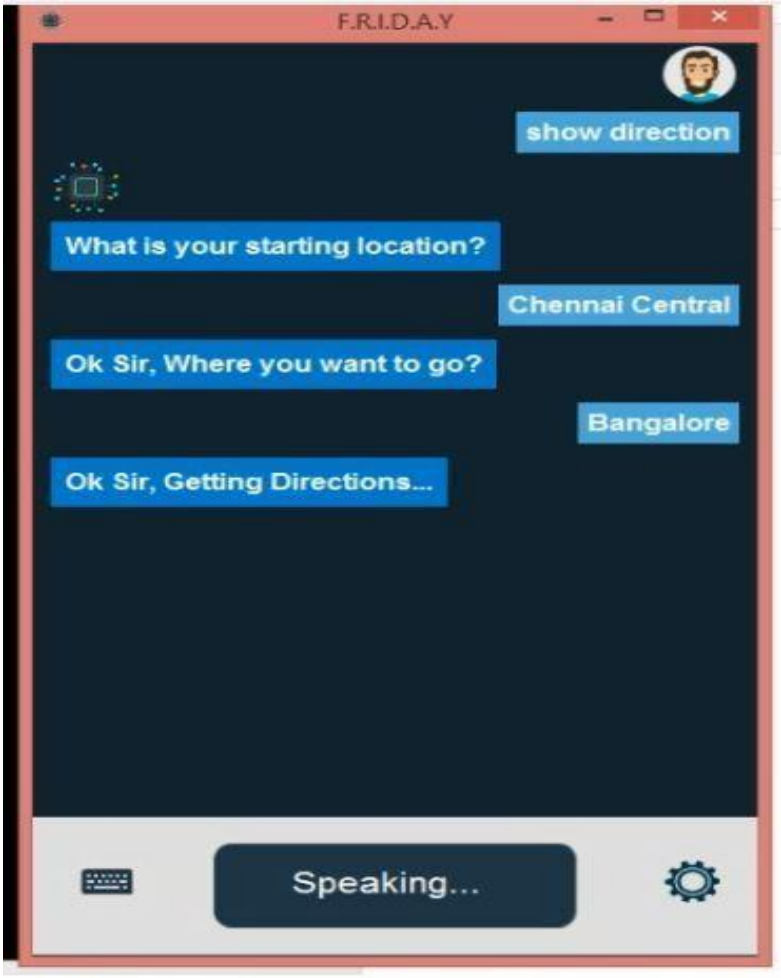


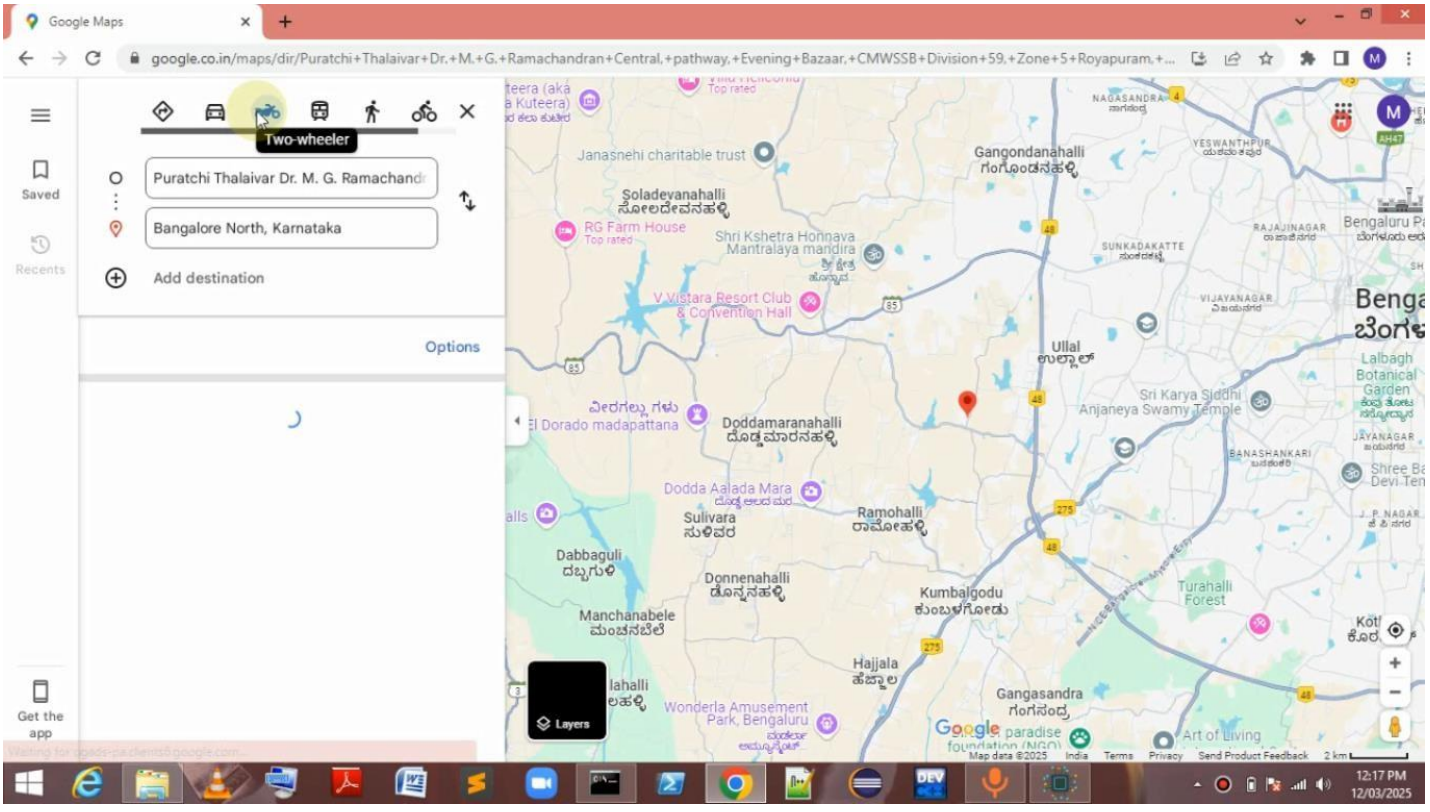

F.R.I.D.A.Y
who is Abdul Kalam
Searching...
Avul Pakir Jainulabdeen Abdul Kalam (; 15 October 1931 – 27 July 2015) was an Indian aerospace scientist and statesman who served as the president of India from 2002 to 2007.
Born and raised in a Muslim family in Rameswaram, Tamil Nadu, he studied physics and aerospace engineering.
Speaking...


F.R.I.D.A.Y
show the images of Abdul Kalam
Here are the images...
Dr. A.P.J. Abdul Kalam
Listening...

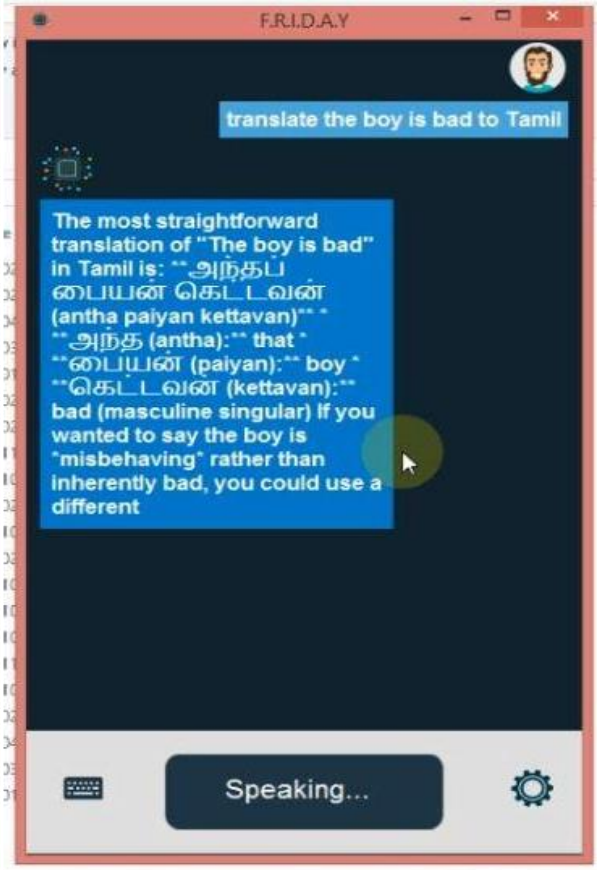

F.R.I.D.A.Y
open Notepad
Listening...

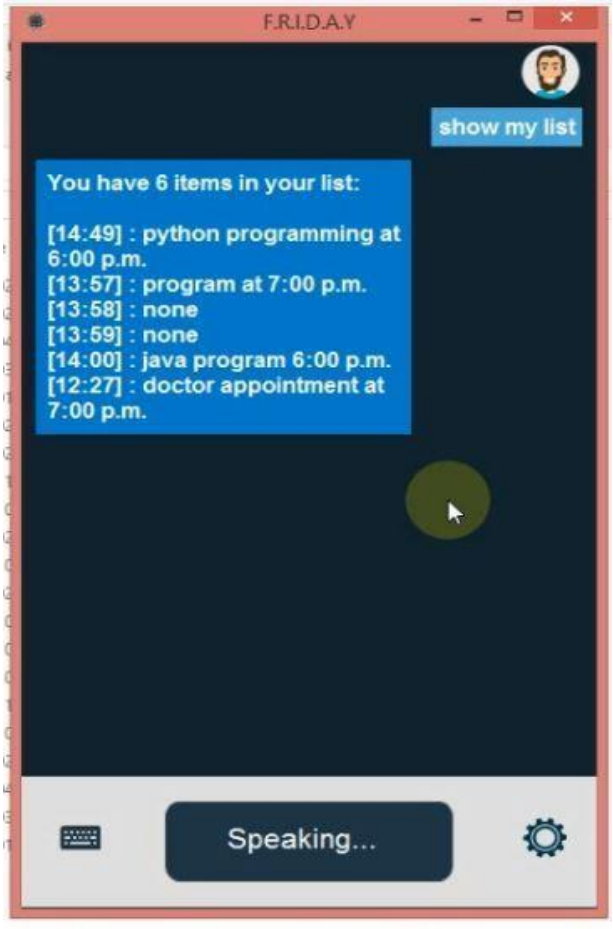

python GUIAS
F.R.I.D.A.Y
Untitled - Notepad
File Edit Format View Help
Listening...

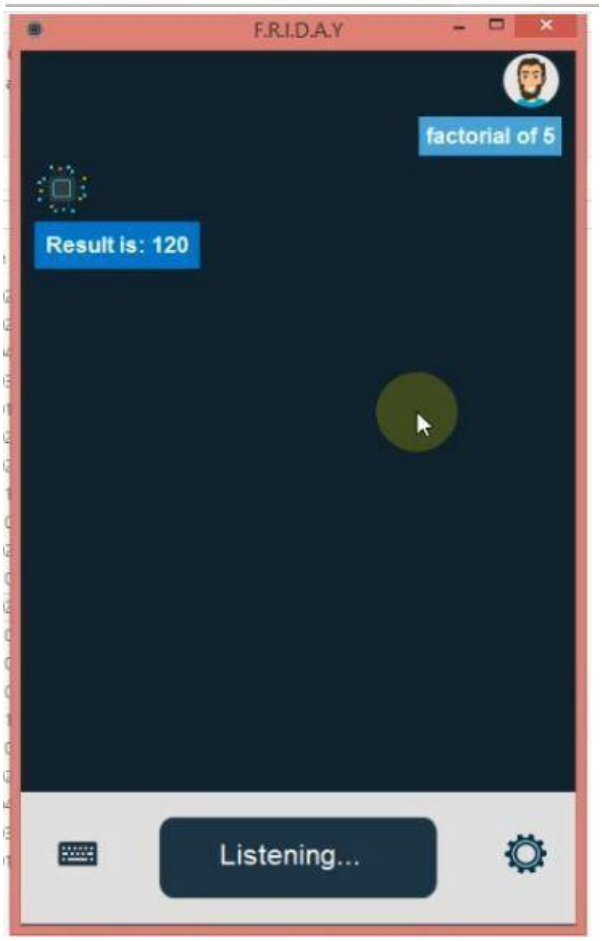

F.R.I.D.A.Y
show direction
What is your starting location?
Chennai Central
Ok Sir, Where you want to go?
Bangalore
Ok Sir, Getting Directions...
Speaking...

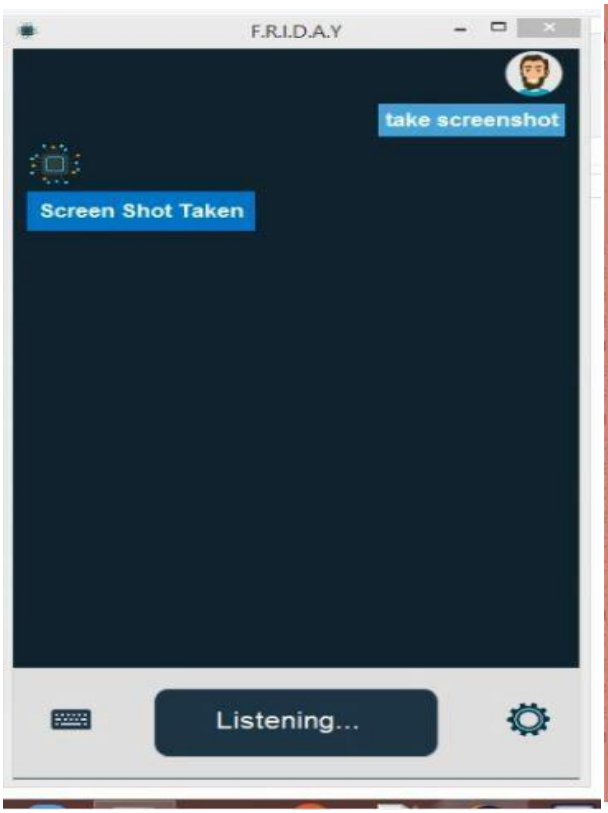

Google Maps
google.co.in/maps/dir/Puratchi+Thalaivar+Dr.+M.+G.+Ramachandran+Central,+pathway,+Evening+Bazaar,+CMWSSB+Division+59,+Zone+5+Royapuram,+...
Two-wheeler
Puratchi Thalaivar Dr. M. G. Ramachandr
Bangalore North, Karnataka
Add destination
Options
Saved
Recents
Get the app
Layers
Janasnehi charitable trust
Gangondanahalli
Soladevanahalli
RG Farm House
Shri Kshetra Honnava Mantralaya mandira
V Vistara Resort Club & Convention Hall
Ullal
Doddamaranahalli
Dodda Aalada Mara
Sulivara
Ramohalli
Dabbaguli
Donnenahalli
Kumbalgodu
Manchanabele
Hajjala
Gangasandra
Wonderla Amusement Park, Bengaluru
Sri Karya Siddhi Anjaneya Swamy Temple
Lalbagh Botanical Garden
Turahalli Forest
Art of Living
Map data ©2025
India
Terms
Privacy
Send Product Feedback
2 km
12:17 PM
12/03/2025


F.R.I.D.A.Y
create a word file
File is created.
Now you can edit this file
Microsoft
Microsoft Office Word 2007
© 2006 Microsoft Corporation. All rights reserved.
Speaking...

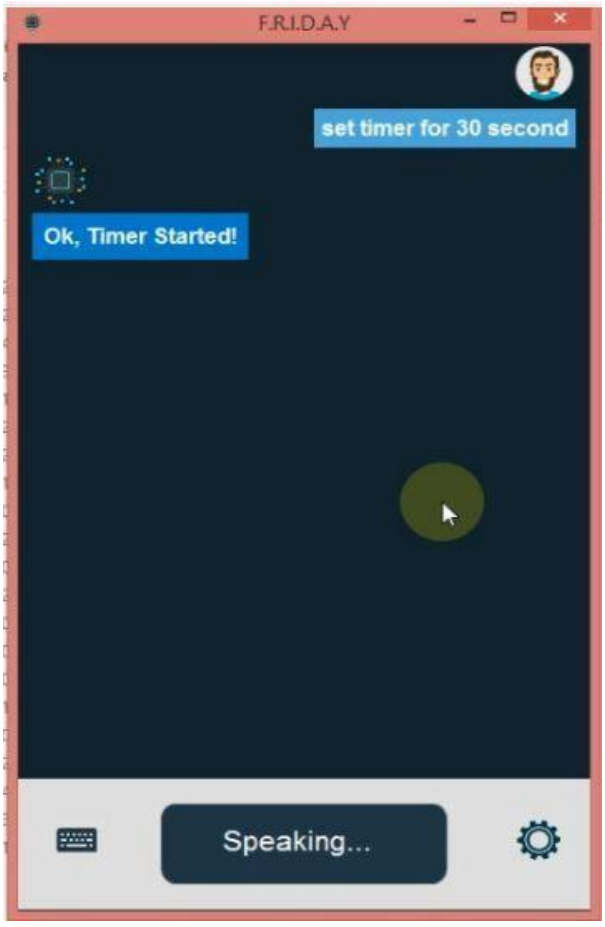

F.R.I.D.A.Y
translate the boy is bad to Tamil
The most straightforward translation of "The boy is bad" in Tamil is: **அந்தப் பையன் கெட்டவன் (antha paiyan kettavan)** * **அந்த (antha):** that * **பையன் (paiyan):** boy * **கெட்டவன் (kettavan):** bad (masculine singular) If you wanted to say the boy is "misbehaving" rather than inherently bad, you could use a different
Speaking...

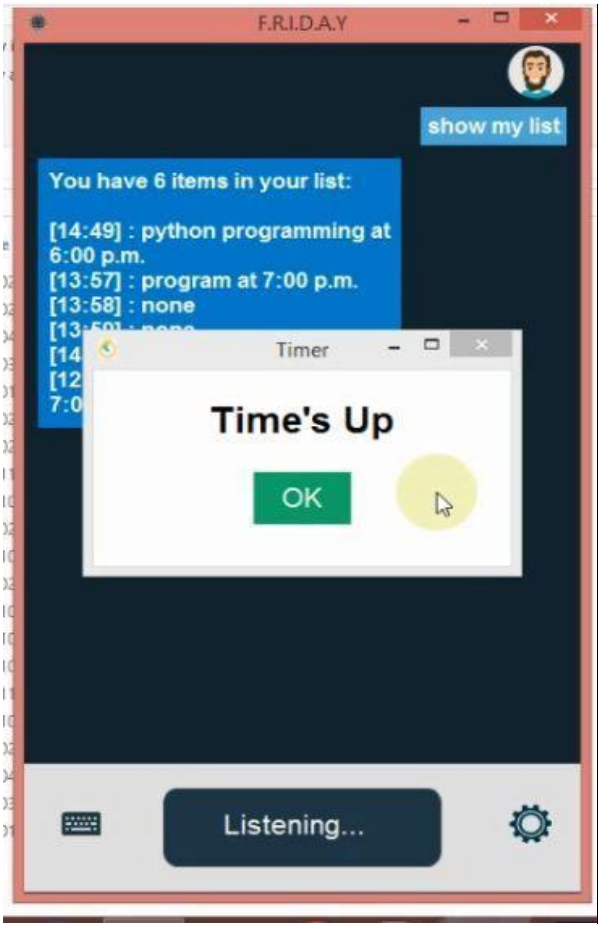

F.R.I.D.A.Y
show my list
You have 6 items in your list:
[14:49] : python programming at 6:00 p.m.
[13:57] : program at 7:00 p.m.
[13:58] : none
[13:59] : none
[14:00] : java program 6:00 p.m.
[12:27] : doctor appointment at 7:00 p.m.
Speaking...

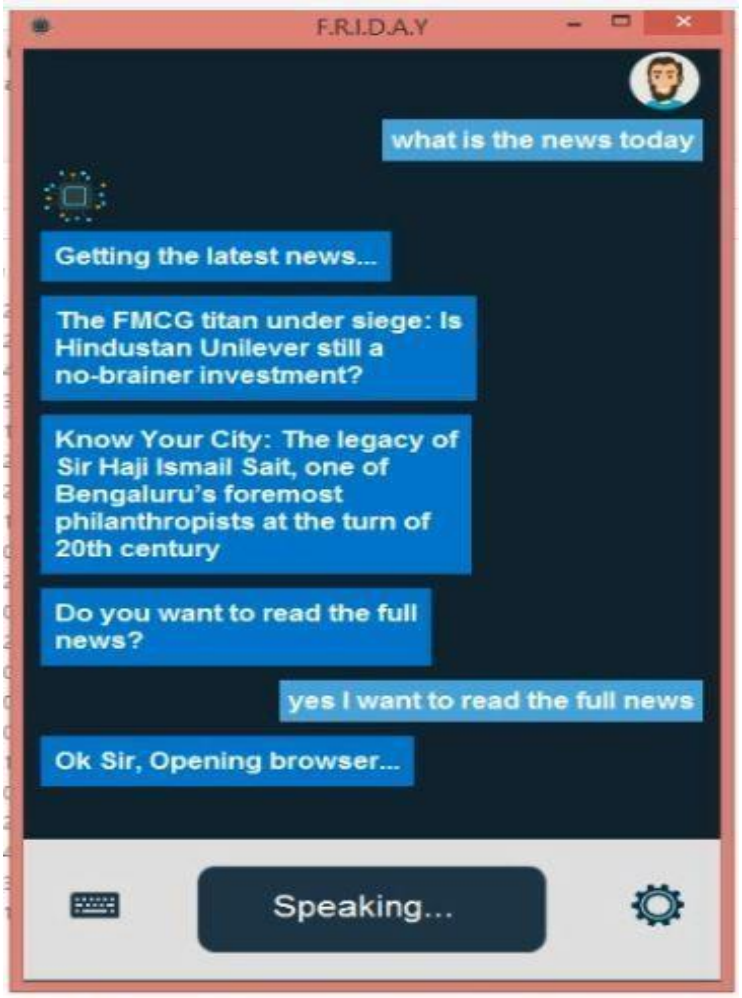

F.R.I.D.A.Y
factorial of 5
Result is: 120
Listening...

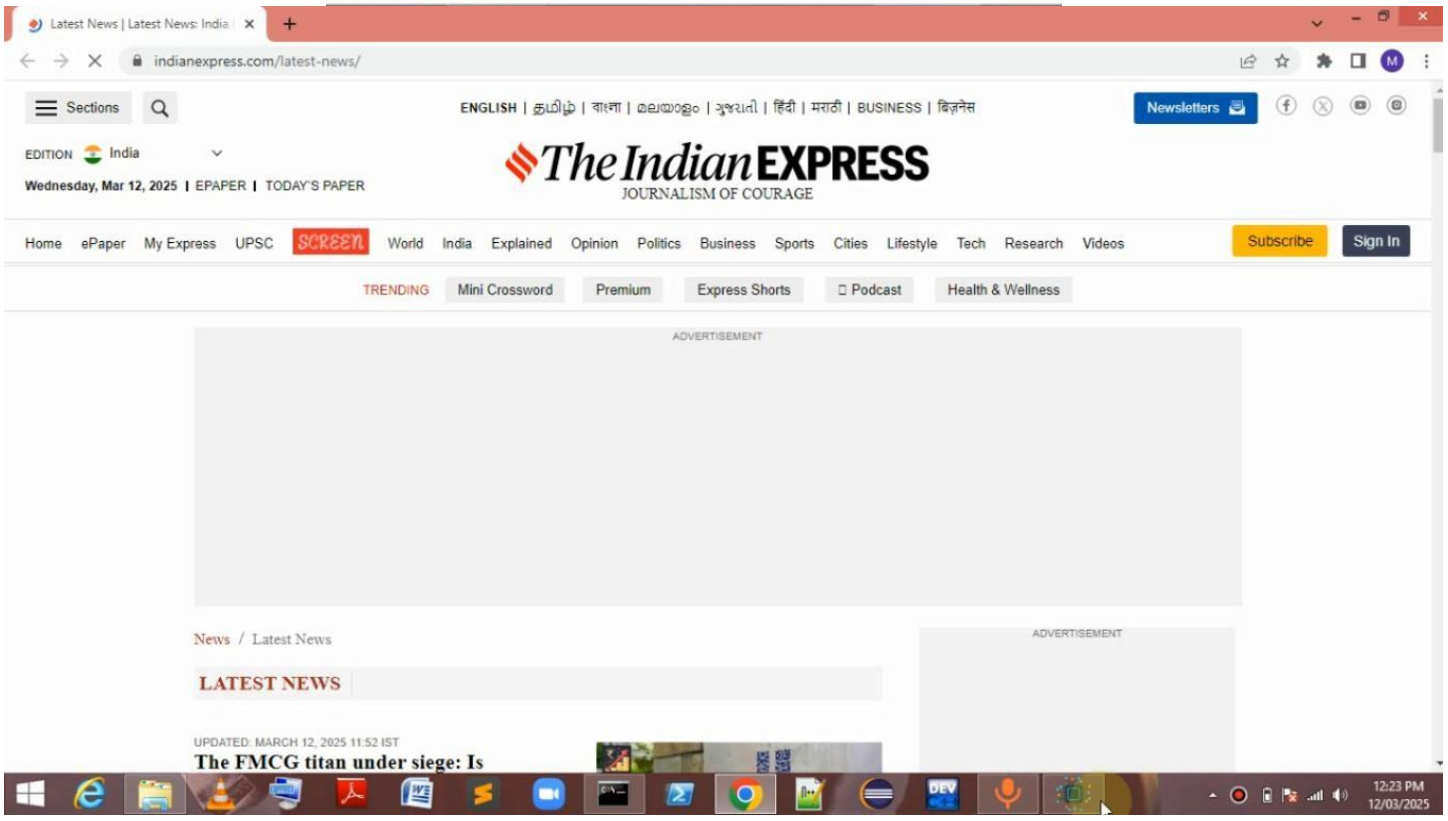

F.R.I.D.A.Y
take screenshot
Screen Shot Taken
Listening...


Rock Paper Scissor
Total Score
0 | 0
Waiting...
Listening...

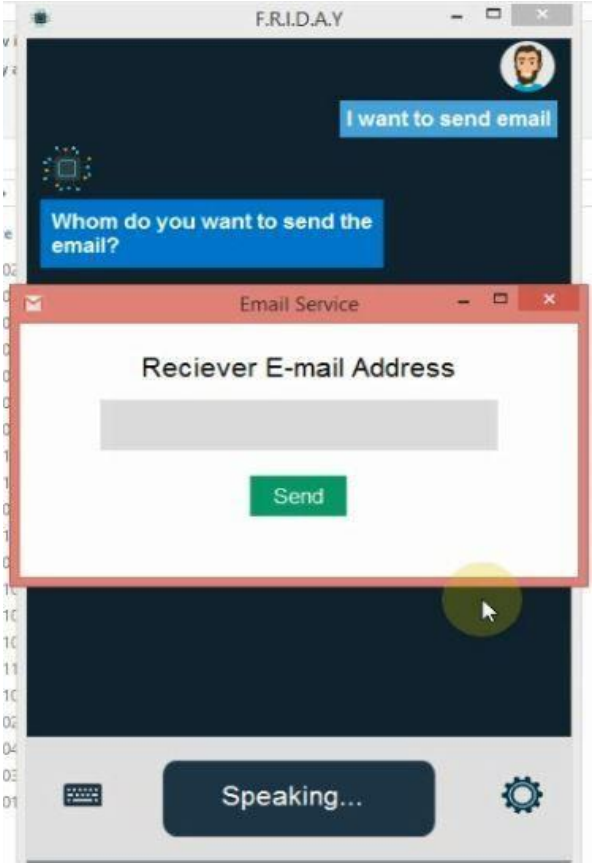

F.R.I.D.A.Y
set timer for 30 second
Ok, Timer Started!
Speaking...

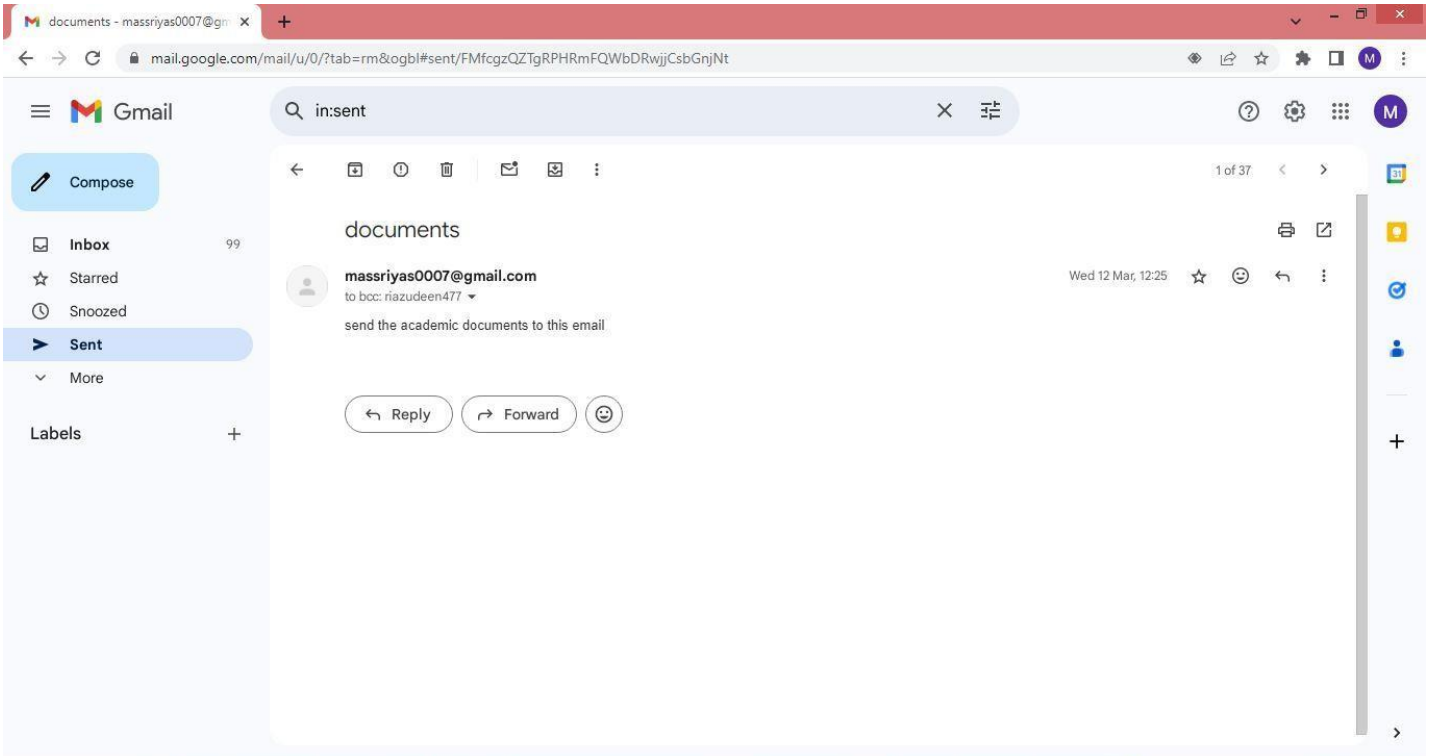

F.R.I.D.A.Y
show my list
You have 6 items in your list:
[14:49] : python programming at 6:00 p.m.
[13:57] : program at 7:00 p.m.
[13:58] : none
Timer
Time's Up
OK
Listening...


F.R.I.D.A.Y
what is the news today
Getting the latest news...
The FMCG titan under siege: Is Hindustan Unilever still a no-brainer investment?
Know Your City: The legacy of Sir Haji Ismail Sait, one of Bengaluru's foremost philanthropists at the turn of 20th century
Do you want to read the full news?
yes I want to read the full news
Ok Sir, Opening browser...
Speaking...

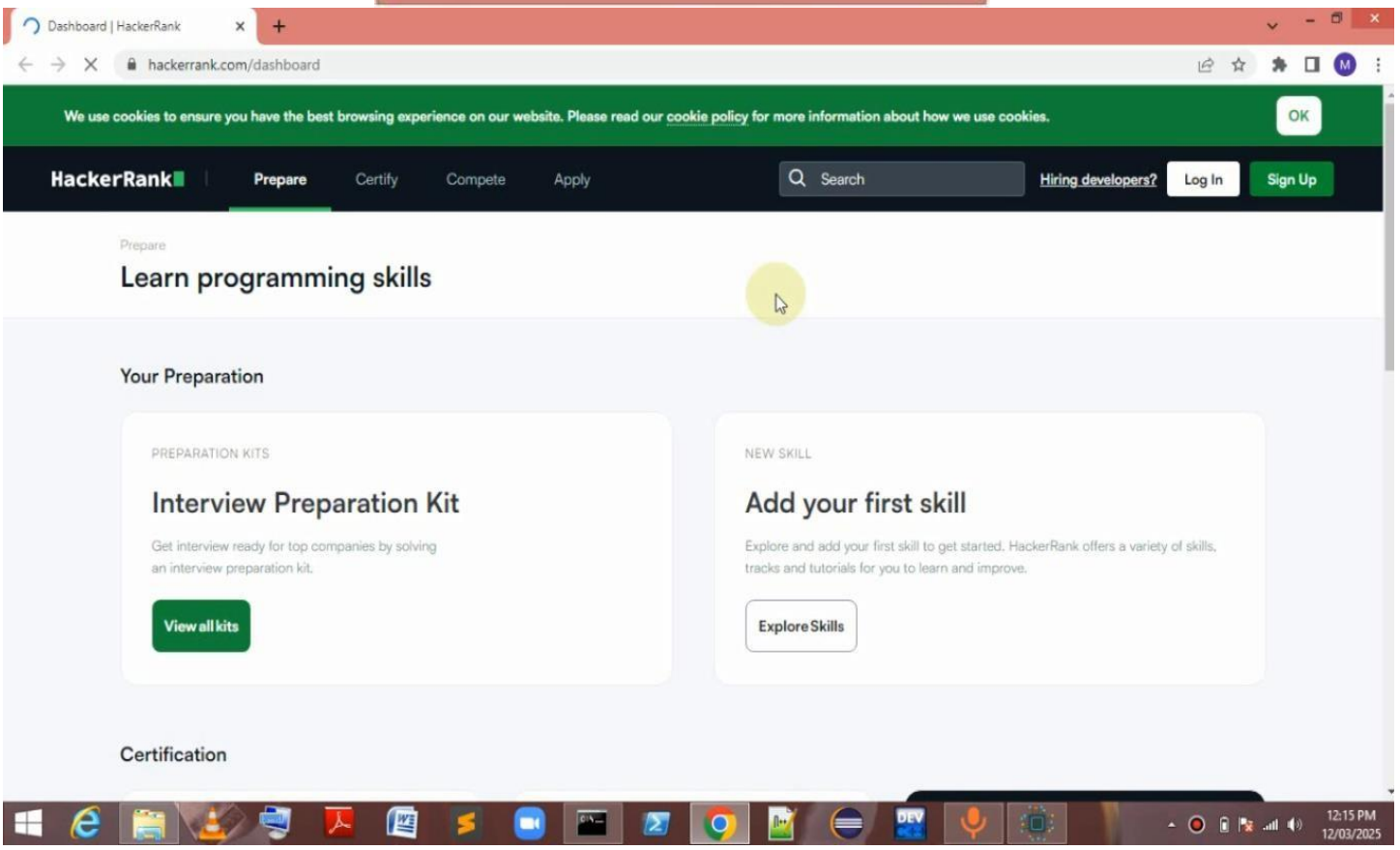

Latest News | Latest News: India
indianexpress.com/latest-news/
Sections
ENGLISH | தமிழ் | বাংলা | മലയാളം | ગુજરાતી | हिंदी | मराठी | BUSINESS | ਪੰਜਾਬੀ
Newsletters
EDITION India
Wednesday, Mar 12, 2025 | EPAPER | TODAY'S PAPER
The Indian EXPRESS
JOURNALISM OF COURAGE
Home ePaper My Express UPSC SCREEN World India Explained Opinion Politics Business Sports Cities Lifestyle Tech Research Videos
Subscribe
Sign In
TRENDING Mini Crossword Premium Express Shorts Podcast Health & Wellness
ADVERTISEMENT
News / Latest News
LATEST NEWS
UPDATED: MARCH 12, 2025 11:52 IST
The FMCG titan under siege: Is
ADVERTISEMENT
12:23 PM
12/03/2025

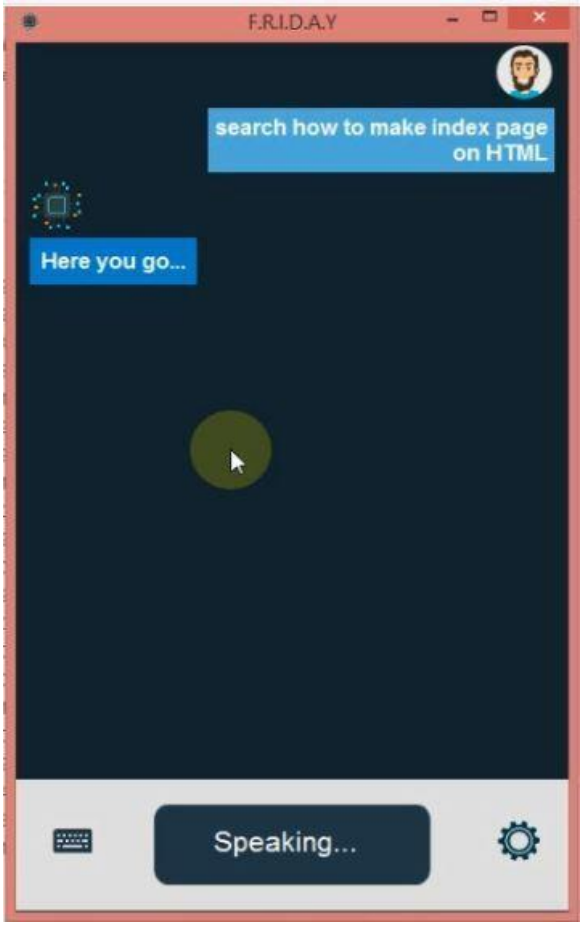

F.R.I.D.A.Y
I want to send email
Whom do you want to send the email?
riazudeen477@gmail.com
What is the Subject?
documents
What message you want to send ?
send the academic documents to this email
Email has been Sent
Listening...

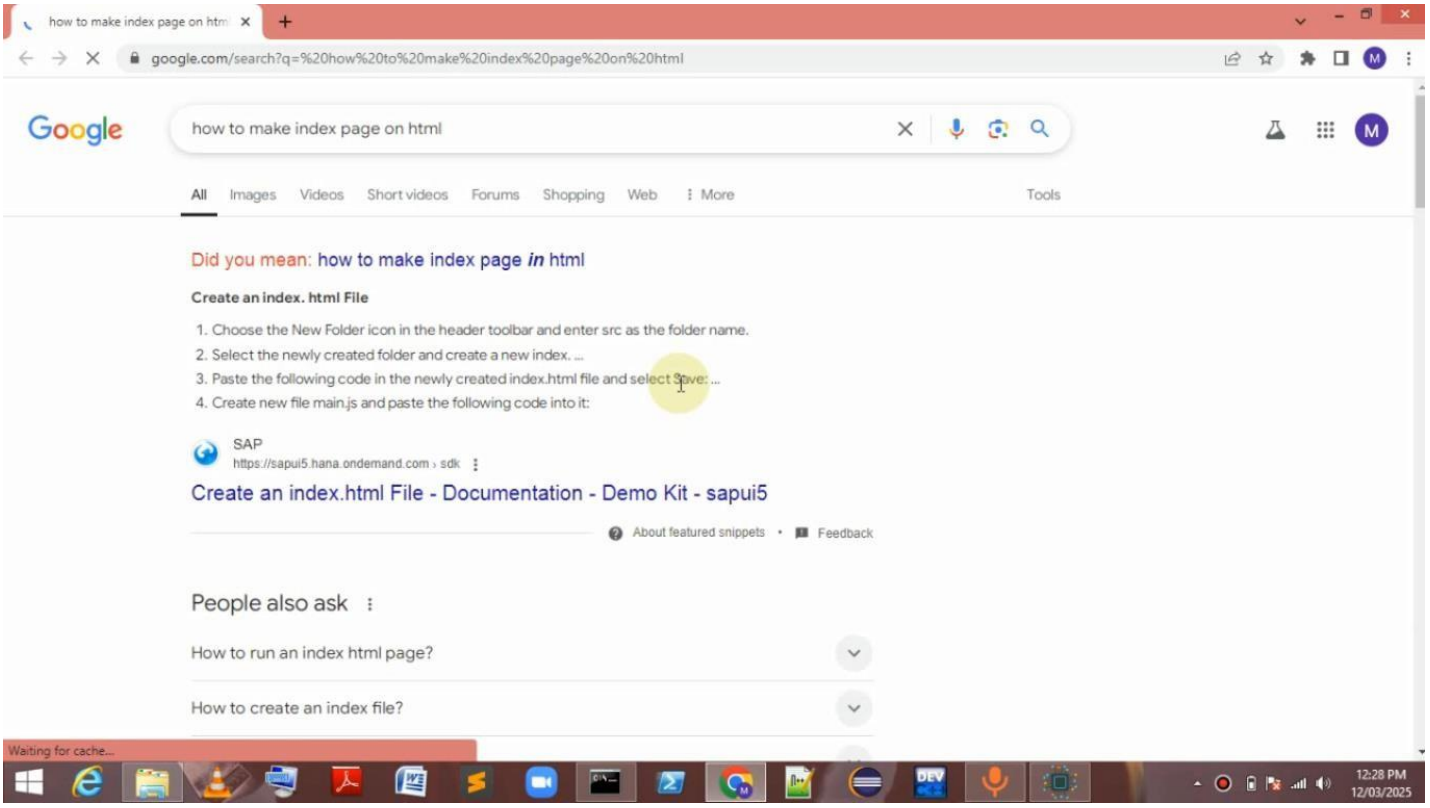

F.R.I.D.A.Y
I want to send email
Whom do you want to send the email?
Email Service
Reciever E-mail Address
Send
Speaking...


documents - massriyas0007@gm
mail.google.com/mail/u/0/?tab=rm&ogbl#sent/FMfcgzQZTgRPHRmFQWbDRwjjCsbGnjNt
Gmail
in:sent
Compose
Inbox 99
Starred
Snoozed
Sent
More
Labels
1 of 37
documents
massriyas0007@gmail.com
to bcc: riazudeen477
Wed 12 Mar, 12:25
send the academic documents to this email
Reply
Forward

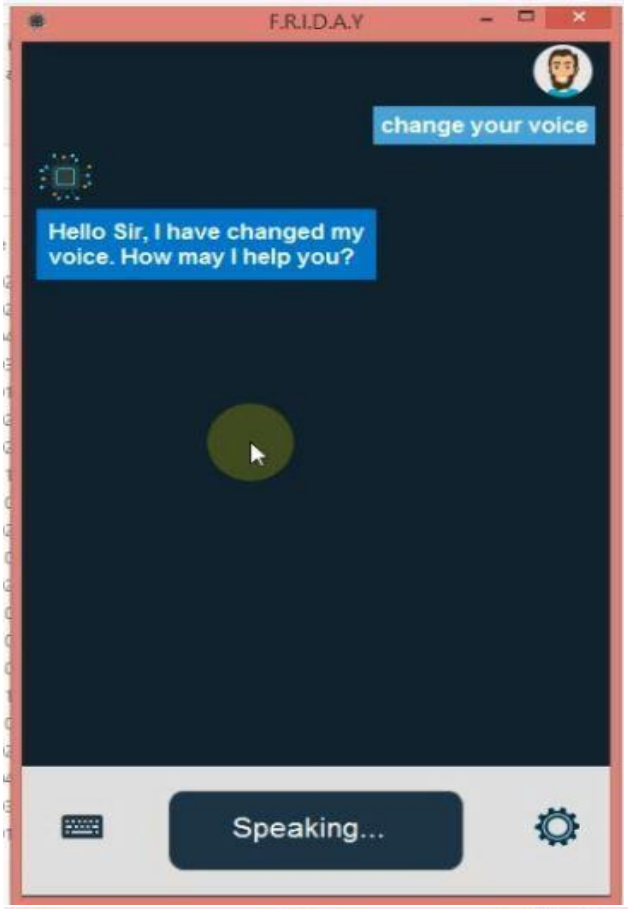

F.R.I.D.A.Y
open hackerrank
Listening...

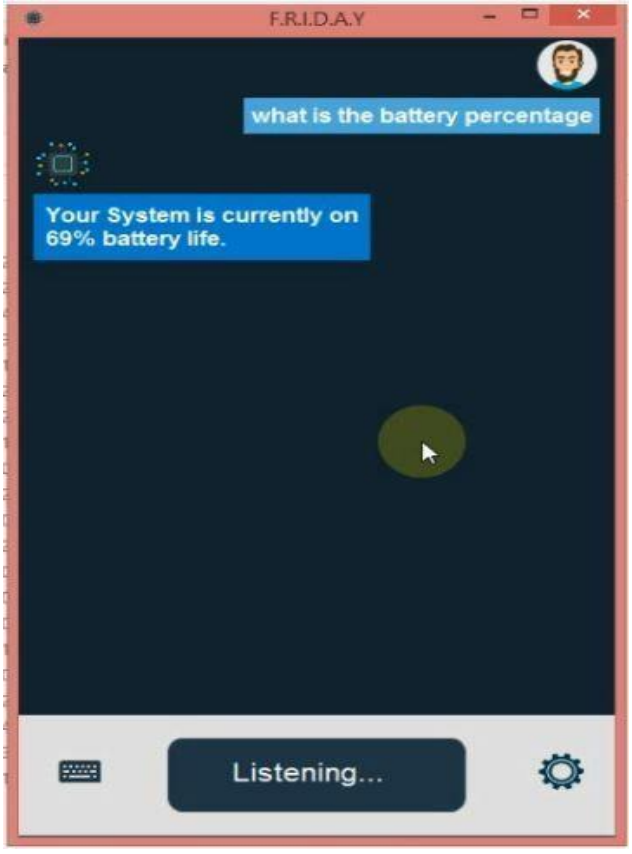

Dashboard | HackerRank
hackerrank.com/dashboard
We use cookies to ensure you have the best browsing experience on our website. Please read our cookie policy for more information about how we use cookies.
OK
HackerRank
Prepare
Certify
Compete
Apply
Search
Hiring developers?
Log In
Sign Up
Prepare
Learn programming skills
Your Preparation
PREPARATION KITS
Interview Preparation Kit
Get interview ready for top companies by solving an interview preparation kit.
View all kits
NEW SKILL
Add your first skill
Explore and add your first skill to get started. HackerRank offers a variety of skills, tracks and tutorials for you to learn and improve.
Explore Skills
Certification
12:15 PM
12/03/2025


F.R.I.D.A.Y
search how to make index page on HTML
Here you go...
Speaking...

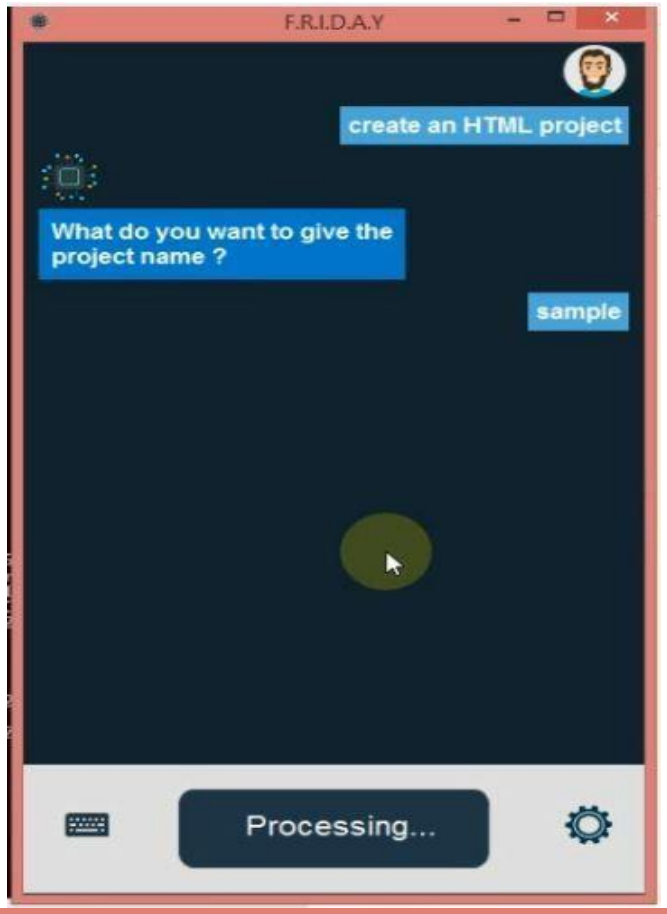

how to make index page on html
google.com/search?q=%20how%20to%20make%20index%20page%20on%20html
Google
how to make index page on html
All Images Videos Short videos Forums Shopping Web More Tools
Did you mean: how to make index page in html
Create an index. html File
1. Choose the New Folder icon in the header toolbar and enter src as the folder name.
2. Select the newly created folder and create a new index. ...
3. Paste the following code in the newly created index.html file and select Save: ...
4. Create new file main.js and paste the following code into it:
SAP
https://sapui5.hana.ondemand.com › sdk
Create an index.html File - Documentation - Demo Kit - sapui5
About featured snippets • Feedback
People also ask
How to run an index html page?
How to create an index file?
Waiting for cache...
12:28 PM
12/03/2025

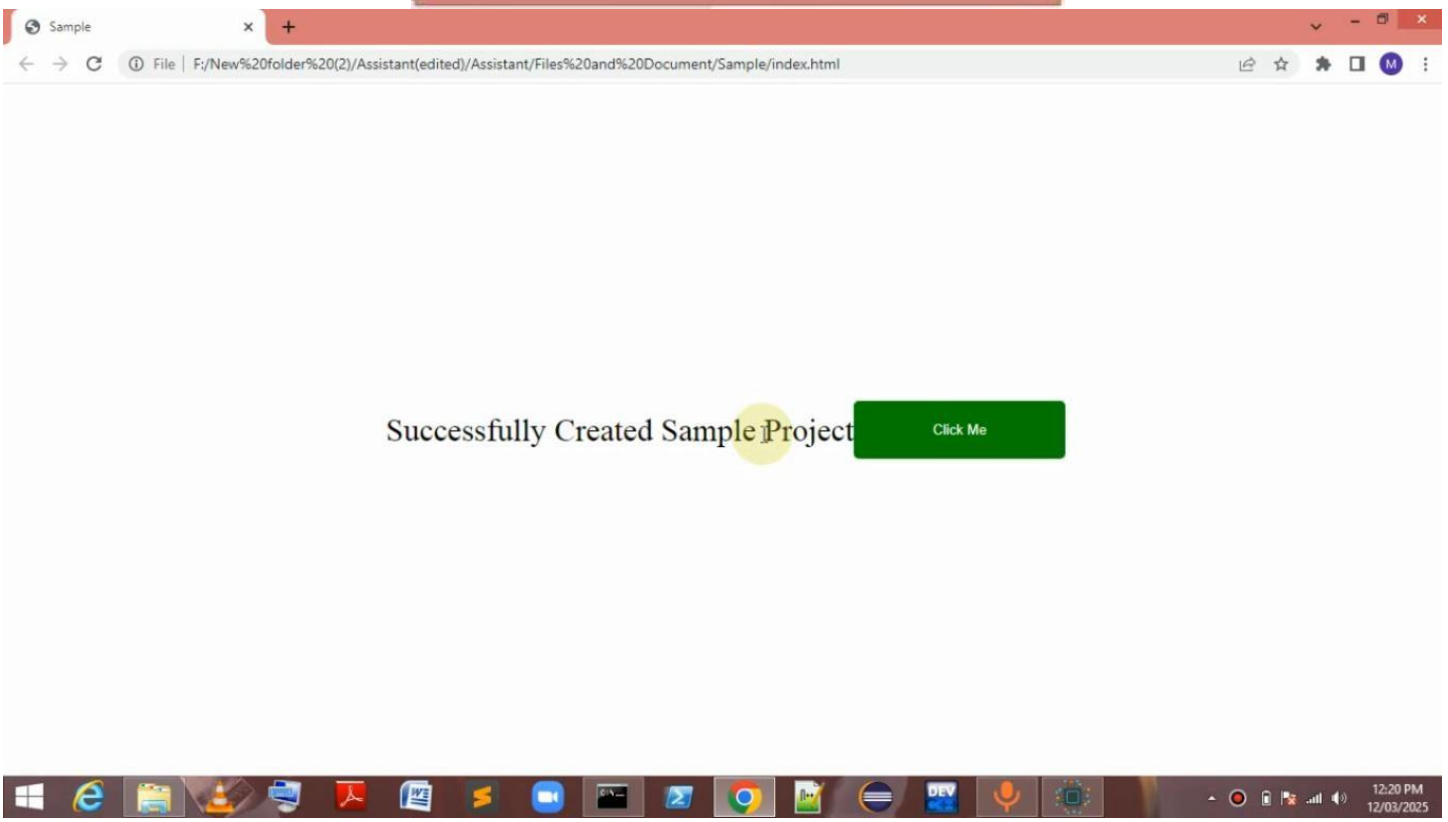

F.R.I.D.A.Y
what is the weather today
Currenlty
29 °C
Wednesday
Town Hall
Listening...


F.R.I.D.A.Y
change your voice
Hello Sir, I have changed my voice. How may I help you?
Speaking...


F.R.I.D.A.Y
what is the battery percentage
Your System is currently on 69% battery life.
Listening...


F.R.I.D.A.Y
give system information
Here is your System Information...
Total Disk Space: 63.71 GB
Free Disk Space: 5.65 GB
Manufacturer: Acer
Model: Aspire E1-431
Owner: jainul
Number of Processors: 1
System Type: x64-based PC
Listening...


F.R.I.D.A.Y
create an HTML project
What do you want to give the project name ?
sample
Processing...


Sample
File | F:/New%20folder%20(2)/Assistant(edited)/Assistant/Files%20and%20Document/Sample/index.html
Successfully Created Sample Project
Click Me
12:20 PM
12/03/2025