VIRTUAL ASSISTANT FOR DESKTOP WITH FACE LOCK

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in Computer Science and Engineering

By

CHILAKAM DHEERAJ REDDY (Reg.No - 39110228)
CHINTHA SURYA SIMHA REDDY (Reg.No – 39110233)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade “A” by NAAC | 12B Status by UGC | Approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI - 600119

APRIL - 2023

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work **CHILAKAM DHEERAJ REDDY(Reg.NO:39110228) AND CHINTHA SURYA SIMHA REDDY (REG NO:39110233)** who carried out the Project Phase-2 entitled **"PERSONAL ASSISTANT FOR DESKTOP WITH FACE LOCK"** under my supervision from January 2023 to April 2023.

S.Nithya

Internal Guide

Ms. NITHYA S, M.E.

[illegible]

Head of the Department

Dr. L. LAKSHMANAN, M.E., Ph.D.

Submitted for Viva voce Examination held on________________________

[illegible]

Internal Examiner

[illegible]

External Examiner

DECLARATION

I, **CHILAKAM DHEERAJ REDDY(Reg.NO:39110228),** hereby declare that the Project Phase-2 Report entitled **"PERSONAL ASSISTANT FOR DESKTOP WITH FACE LOCK"** done by me under the guidance of **Ms. NITHYA S, M.E.** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.

Dheeraj

DATE:

PLACE: Chennai **SIGNATURE OF THE CANDIDATE**

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph. D**, **Dean**, School of Computing, **Dr. L. Lakshmanan M.E., Ph.D.,** Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Ms. NITHYA S, M.E.,** for Her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my phase-2 project work.

I wish to express my thanks toall Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTARCT

In this modern era day to day life became smarter and interlinked with technology. A Personal Virtual Assistant allows a user to simply ask questions in the same manner that they would address a human, and are even capable of doing some basic tasks like opening apps, reading out news, taking notes etc., with just a voice command.

The voice assistance takes the voice input through our microphone and it converts our voice into computer understandable language gives the required solutions which are asked by the user. Natural language processing algorithm helps computer machines to engage in communication using natural human language in many forms.

This system is designed to be used efficiently on desktops. Personal assistant software improves user productivity by managing routine tasks of the user and by providing information from online sources to the user. It is effortless to use

This engages the ability to communicate socially through natural language processing, holding (and analysing) information within the context of the user. It is suggested that new technologies may soon make the idea of virtual personal assistants a reality.

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

The concept of a smart assistant has become widely known and popular over the last decade. Commercial devices such as Amazon Alexa, Google Home, and Mycroft can interact with users through speech recognition and synthesis, provide a variety of network-based services, and can interface with smart home automation systems, providing them with an advanced user interface.

The availability of many network services and an increasing number of additional skills, or capabilities that can be easily added to smart assistants is driving the spread of such speech-enabled smart assistants. Their potential is still limited, however, by their inability to extract real-time visual information from video data, either concerning the user or the environment Spoken dialogue systems are intelligent agents that can assist users in completing tasks more efficiently through spoken interactions.

Personal assistants, also known as virtual personal assistants, intelligent personal assistants, digital personal assistants, or voice assistants, are devices that help people. Personal assistant agents are a new type of software that acts on the user's behalf to find and filter information, negotiate for services, easily automate complex tasks, and collaborate with other software agents to solve complex problems. If the laptop/desktop can learn and adapt to the user's behavior, this can be developed. The laptop/desktop must collect training data from a user's daily activities and apply machine learning techniques to the data.

The model that is created would be able to predict the behavior ahead of time such features in a laptop/desktop could make life easier for the user, such as notifications based on location rather than time. One of the most studied and popular was the direction of interaction, based on the understanding of the machine by the machine of the natural human language. It is no longer a human who learns to communicate with a machine, but a machine learns to communicate with a human, exploring his actions, habits, behavior and trying to become his personalized assistant.

Such features in a laptop/desktop could make life easier for the user, such as notifications based on location rather than time. Time-based reminders are more popular and static, but they are inconvenient to use. It will show your current position as well as recognize your face. The programmers that run on your desktop and laptop must make effective use of resources in order to avoid unnecessary power consumption.
Voice searches have dominated over text search. Web searches conducted via mobile devices have only just overtaken those carried out using a computer and the analysts are already predicting that 50% of searches will be via voice by 2020.Virtual assistants are turning out to be smarter than ever. Allow your intelligent assistant to make email work for you. Detect intent, pick out important information, automate processes, and deliver personalized responses.

This project was started on the premise that there is enough openly available data and information on the web that can be utilized to build a virtual assistant that has access to making intelligent decisions for routine user activities.

CHAPTER 2

LITERATURE SURVEY

Voice assistant consume stood the hot topic of AI in fresh studies. We collect several papers in order to examine voice assistant and then analyze data.

P. meliorate [1]. A computer mainly grounded method for execution a command through a voice user interface on a subdivision of items. The item contains written contents that is converted to voice output.

A.M Weeretunga [2]. This project is mainly focused to help visually disabled to access social media and other internetbased services, because understanding digital content is an extremely important and hard problem for this user group.

Prajyot Mane [3]. The system enables the user to get fractures provided by different applications on a single platform. The application will world and provide profile management automation. This system performs statically without any human intervention.

Veton Kepuska [4]. The most familiar application of phone is “SIRI” which reasons the end user to inform end user adaptablewith voice and it furthermore respond to the voice charge of the client in this scheme open data is assembly consideration for imaginative administrative creation, mainly in the region ancient government, bio science and attached undertaking.

Schlash s [5]. This paper presents a voice associate which apply exposed information as its education foundation. It is highlighted by alteration of exactness as per the user reproaches and procurement of unregistered statistics by the user provision.

SureshV.Reddy[6]. This paper presents about the design and implementation of Virtual Voice Assistance and understanding design content is an extremely important and hard problem for this user group.

Purushotham Botla [7]. This paper presents about the design and implementation of Virtual Voice Assistance and Designing Personal Assistant Software for Task Management using Semantic Web Technologies and Knowledge Databases.

Abhijeet Thakur [8]. When the computer system first appeared, it was a pipe dream to have full-fledged interaction with the machine. he considered the future capabilities of voice recognition and facial detection. He gave an individual can access the system using facial recognition, and face detection helps to secure the data by ensuring that no other person can access the system. It employs machine learning algorithms and assists users in gaining secure access.

Mayura D Tapkire [9].

This paper presents about the design and implementation of virtiual assistants and the understanding of the device means AI needs to be integrated with the device so that the device can work in a smart way and can also control IoT and devices and can also respond to query which will search the web for results and process it.

Dhiraj prathap singh [10]. In this paper we have discussed a Voice Activated Personal Assistant developed using python. This assistant currently works online and performs basic tasks like weather updates, stream music, search Wikipedia, open desktop applications, etc.

2.1 INFERENCES FROM LITREATURE SURVEY

Cortana can likewise read your messages, track your area, watch your perusing history, check your contact list, watch out for your date-book, and set up this information together to propose valuable data, on the off chance that you enable it. You can likewise type your inquiries or solicitations, if you want to not stand up uproarious. It is only desktop based virtual assistant.

Siri: Siri has been an integral part of iOS since the dispatch of iOS 5 of every 2011. It began with the nuts and bolts, for example, climate and informing, yet has extended significantly from that point forward to help more outsider mix with MacOS. While Siri’s jokes are unbelievable, the virtual aide is getting more able consistently. Presently, you can request that it call individuals, send messages, plan gatherings, dispatch applications and recreations, and play music, answer questions, set updates, and give climate conjectures

Google Assistant: Google Assistant (which has consolidated capacities from the more seasoned Google now, as now is being eliminated) is unique in relation to Cortana and Siri.

Alexa: While sharing different features similarly as various VAs, Alexa is in its own one-of-a-kind class. Amazon’s voice partner is not centred on portable or PC purposes, but instead for the independent Amazon Echo speaker and a set number of Amazon Fire gadgets, with a more prominent focus on entire house administration and administrations as opposed to PC situated errands.

Each business visionary, side trickster and multitasking proficient out there would love to have a virtual assistant right hand to go up against a portion of the dull every day errands that accompany existing in the advanced time. Similarly, as with any developing innovation, in any case, it can be hard to isolate the build up from the certainties.

There are four noteworthy players seeking consideration: Amazon (Alexa), Apple (Siri), (Google Assistant) and Microsoft (Cortana). I invested hours testing each of the four assistants by making inquiries and giving charges that numerous business clients would utilize.

Amid the testing procedure, I noticed the accomplishment of the AI’s reaction to me, and in addition different components a planned users may think about, for example, simplicity of setup, general capacity to perceive my voice and relevant comprehension.

2.2 OPEN PROBLEMS AND EXISTING PROBLEMS

Personal Assistant systems have become an integral part of our daily lives. With the increasing use of smart devices and the growing demand for efficient and personalized services, the development of Personal Assistant systems has gained significant attention in recent years.

A Personal Assistant system is designed to perform various tasks, such as setting reminders, scheduling appointments, making phone calls, sending messages, and even controlling smart home devices. Python is one of the popular programming languages used to build Personal Assistant systems due to its simplicity and flexibility.

Despite the recent advancements in Personal Assistant systems, there are still several open problems that need to be addressed. In this paper, we will discuss three main open problems in the existing system for the project Personal Assistant using Python.

PERSONALISATION

Personalization is a crucial aspect of Personal Assistant systems. The system must be able to understand the user's preferences and behavior to provide personalized services. However, most existing Personal Assistant systems have limited personalization capabilities. For instance, the system may only provide generic responses to user queries, regardless of the user's context and preferences.

To address this open problem, researchers can explore machine learning and natural language processing techniques to enable Personal Assistant systems to understand the user's context and preferences better. This can be achieved by analyzing the user's search history, social media activity, and other relevant data sources to gain insights into the user's interests and behavior. Additionally, Personal Assistant systems can leverage user feedback to improve personalization.

INTEGRATION AND THIRD-PARTY SERVICES

Personal Assistant systems should be able to integrate seamlessly with third-party services to provide users with a more comprehensive experience. However, integration with third-party services can be challenging due to the lack of standardization and the varying interfaces of these services. Furthermore, security and privacy concerns must be addressed when integrating with third-party services.

To address this open problem, researchers can explore the use of APIs and webhooks to facilitate integration with third-party services. Additionally, standards and protocols can be developed to ensure consistency and interoperability across different services. To address security and privacy concerns, Personal Assistant systems can implement robust authentication and authorization mechanisms and adhere to data protection regulations.

MULTI-MODEL INTERACTION

Most existing Personal Assistant systems rely solely on voice-based interaction. However, this approach can be limiting for users with speech impairments or in environments where voice-based interaction is not feasible. Additionally, voice-based interaction may not be suitable for certain tasks, such as reading and browsing.

To address this open problem, researchers can explore the use of multimodal interaction, which combines multiple modes of input, such as voice, touch, and gestures, to provide a more natural and flexible interaction experience. Additionally, Personal Assistant systems can leverage computer vision and image recognition technologies to enable users to interact with the system through visual cues and gestures.

ACCURACY ON SPEECH RECOGNITION

One of the main challenges in developing a personal assistant application is achieving accurate speech recognition. Speech recognition is the process of converting human speech into text that can be processed by a computer. It is a critical component of any personal assistant application that involves voice commands.

The accuracy of speech recognition is affected by various factors such as background noise, accent, and speech rate. While there have been significant improvements in speech recognition technology in recent years, there is still a long way to go to achieve perfect accuracy.

To address this problem, researchers are exploring various approaches such as deep learning algorithms, speech enhancement techniques, and acoustic modeling. These approaches are designed to improve the accuracy of speech recognition by reducing the impact of external factors that affect speech recognition.

In conclusion,
Personal assistant applications have the potential to revolutionize the way people manage their daily tasks. However, there are still several open problems that need to be addressed to make these applications more effective and useful. In this document, we discussed three open problems in existing systems for personal assistants using Python: accuracy of speech recognition, personalization and context awareness, and integration with multiple devices and services.
Researchers are exploring various approaches to address these problems, and we can expect significant progress in the development of personal assistant applications in the coming years.
Personal Assistant systems using Python have several open problems that need to be addressed to improve their performance and usability. Personalization, integration with third-party services, and multimodal interaction are some of the critical areas where research can be focused to develop more efficient and user-friendly Personal Assistant systems**.**

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDIES/RISK ANALYSIS OF THE PROJECT

Feasibility study can help you determine whether you should proceed with your project. It is essential to evaluate cost and benefit. It is essential to evaluate cost and benefit of the proposed system. Five types of feasibility study are taken into consideration.

3.1.1 Technical feasibility: It includes finding out technologies for the project, both hardware and software. For virtual assistant, user must have microphone to convey their message and a speaker to listen when system speaks. These are very cheap now a days and everyone generally possess them. Besides, system needs internet connection. While using JIA, make sure you have a steady internet connection. It is also not an issue in this era where almost every home or office has Wi-Fi.

3.1.2 Operational feasibility: It is the ease and simplicity of operation of proposed system. System does not require any special skill set for users to operate it. In fact, it is designed to be used by almost everyone. Kids who still do not know to write can read out problems for system and get answers.

3.1.3 Economic feasibility: Here, we find the total cost and benefit of the proposed system over current system. For this project, the main cost is documentation cost. User also would have to pay for microphone and speakers. Again, they are cheap and available. As far as maintenance is concerned, JIA will not cost too much.

3.1.4 Organizational feasibility: This shows the management and organizational structure of the project. This project is not built by a team. The management tasks are all to be carried out by a single person. That will not create any management issues and will increase the feasibility of the project

3.1.5 Cultural feasibility: It deals with compatibility of the project with cultural environment. Virtual assistant is built in accordance with the general culture. The project is named JIA to represent Indian culture without undermining local beliefs.
This project is technically feasible with no external hardware requirements. Also, it is simple in operation and does not cost training or repairs. Overall feasibility study of the project reveals that the goals of the proposed system are achievable. Decision is taken to proceed with the project.

3.2 SOFTWARE AND HARDWARE REQUIREMENTS SPECIFICATION:

3.2.1 SOFTWARE REQUIREMENTS:

Operating System: The desktop application will require a compatible operating system that supports Python and the necessary dependencies. For example, Windows 10, macOS, or Linux distributions such as Ubuntu.

Python Version: The application will require a compatible version of Python programming language to be installed on the desktop computer. The recommended version is Python 3.x, such as Python 3.7 or higher, as it provides the latest features and security updates and is widely used for desktop applications development.

Python Libraries: The following Python modules or libraries may be required for implementing specific functionalities of the personal assistant:

- **Speech Recognition:** A library for converting speech to text, such as SpeechRecognition.
- **Text-to-Speech:** A library for converting text to speech, such as pyttsx3 or gTTS.

- **Natural Language Processing**: Libraries for processing and analyzing natural language, such as NLTK or SpaCy.
- **GUI Development:** Libraries for creating graphical user interfaces (GUI) for the desktop application, such as Tkinter, PyQt, or PyGTK.
- **Web Scraping:** Libraries for extracting data from websites, such as BeautifulSoup or Scrapy.
- **Database:** Libraries for managing data storage and retrieval, such as SQLite, MySQL, or MongoDB, depending on the specific requirements of the personal assistant.
- **PyAudio**: To provide audio input and output capabilities for voice interactions.
- **Requests:** To make HTTP requests for fetching data from online sources.
- **Beautiful Soup:** To parse HTML and extract data from websites.
- **Pandas**: To manipulate and analyze data in tabular form.
- **Matplotlib:** To create visualizations and plots for data analysis.
- **PyQT or Tkinter:** To build the graphical user interface (GUI) for the personal assistant application.

3.2.2 HARDWARE REQUIREMENTS:

The personal assistant project does not have specific hardware requirements, as it is designed to run on a desktop computer. However, the performance of the application may vary depending on the hardware specifications of the computer, such as CPU speed, RAM, and storage. A computer with a reasonably fast processor, sufficient RAM (4 GB or higher recommended), and ample storage space for storing the project files and data is recommended for optimal performance.

Processor: A modern processor with at least dual-core or higher for efficient performance.

Memory & Storage : Sufficient RAM to handle the application's memory requirements, depending on the size of the dataset and complexity of the functionalities. Adequate storage space for the application itself, as well as any data storage requirements, such as databases or other resources.

Microphone and Speaker: If the personal assistant project includes voice recognition and voice output capabilities, a microphone and speaker are required for capturing voice commands and providing audio responses. If the application includes speech recognition and text-to-speech functionalities, the desktop should have a microphone and speaker or headphone for audio input and output.

Internet Connectivity: If the personal assistant project requires online data retrieval or communication with online services, an active internet connection is required for accessing the internet.

Accessibility Requirements: The personal assistant project should comply with accessibility guidelines and standards, such as providing keyboard shortcuts, supporting screen readers, and ensuring the application is usable for people with visual or hearing impairments.

Security Requirements: The personal assistant project should adhere to best practices for securing data and protecting user privacy. This includes storing sensitive information securely, using encryption for communication, and following proper authentication and authorization mechanisms.

User Interface (UI) Requirements: The personal assistant project should have a user-friendly and intuitive graphical user interface (GUI) that allows users to interact with the application easily. The GUI should be responsive, visually appealing, and provide clear feedback to users about the status of the application and the results of their interactions.

Testing Requirements: The personal assistant project should undergo thorough testing to identify and fix any bugs or issues before deployment. This includes unit testing, integration testing, and system testing, as well as testing for different scenarios, input data, and edge cases to ensure the robustness and reliability of the application.

Documentation Requirements: The personal assistant project should be accompanied by comprehensive documentation that includes installation instructions, user guides, technical documentation, and code documentation. The documentation should be clear, concise, and easy to understand, providing sufficient information for users and developers to use, modify, and maintain the application.

Licensing Requirements: The personal assistant project should comply with all relevant licensing requirements, including open-source licenses for any third-party libraries or modules used in the application. The licenses should be properly documented and acknowledged in the project's documentation and source code.

3.3 SYSTEM USE CASE

Introduction to System Use Case for Personal Assistant using Python Modules
System Use Case is a methodology used in software development to define the functional requirements of a system. A use case represents a specific interaction between the user and the system, which describes the flow of events that occur when a user performs a specific task.

In the context of developing a personal assistant using Python modules, system use case can be used to define the various use cases or tasks that the personal assistant application should be able to perform. This can include tasks such as setting reminders, scheduling appointments, sending emails, making phone calls, and performing other tasks on behalf of the user.

In this document, we will discuss three use cases for the personal assistant application using Python modules.

3.3.1 USE CASE 1: PLAYING MUSIC

Actor: User

Preconditions:
The personal assistant application is installed and running on the user's device.
The user has granted the application appropriate permissions and access to their music library or streaming service.

Main Flow:
The user asks the personal assistant application to play a specific song or artist by speaking a voice command. The personal assistant application uses a Python module such as Speech Recognition to convert the user's voice command to text.
The personal assistant application uses a Python module such as spotipy or pafy to search for the requested song or artist and retrieve the relevant music data. The personal assistant application uses a Python module such as pygame or vlc to play the retrieved music data. The personal assistant application confirms to the user that the music is playing.

Alternative Flow:
If the personal assistant application is unable to find or play the requested song or artist, it prompts the user to try a different request or provides an error message.

Postconditions:
The personal assistant application has played the requested song or artist for the user.

Extensions:

The user can ask the personal assistant application to play a specific playlist or genre of music, and the personal assistant application can use a Python module such as spotipy or pafy to search for and retrieve the relevant music data. The personal assistant application can be configured to play music from a specific streaming service or music library based on the user's preference.

The personal assistant application can provide additional music-related information such as lyrics, album information, and artist bios using Python modules such as lyricsgenius and lastfm. The personal assistant application can be extended to support voice-controlled playback commands such as play, pause, skip, and volume control using Python modules such as Speech Recognition and pygame.

3.3.2 USE CASE 2: Sending E-Mail

Actor: User

Preconditions:

The personal assistant application is installed and running on the user's device.

The user has granted the application appropriate permissions and access to their email account.

Main Flow:

The user asks the personal assistant application to send an email to a specific recipient by speaking a voice command. The personal assistant application uses a Python module such as Speech Recognition to convert the user's voice command to text. The personal assistant application prompts the user to specify the email subject and body text.

The personal assistant application uses a Python module such as smtplib to authenticate the user's email account and create a secure connection to the email server. The personal assistant application uses a Python module such as email to create an email message with the specified subject and body text and the recipient's email address.

The personal assistant application uses a Python module such as smtplib to send the email message. The personal assistant application confirms to the user that the email has been sent.

Alternative Flow:
If the personal assistant application is unable to send the email, it prompts the user to check their internet connection or contact settings or provides an error message.

Postconditions:
The personal assistant application has sent the email to the specified recipient.

Extensions:
The personal assistant application can be configured to send emails using a specific email provider or service, such as Gmail or Outlook, based on the user's preference.
The personal assistant application can provide a list of recent contacts or frequently contacted contacts for the user to select from if the user does not specify a recipient in the voice command. The personal assistant application can be extended to support attachments such as images, videos, or documents using Python modules such as email and smtplib. The personal assistant application can be extended to support voice-controlled email composition using Python modules such as Speech Recognition and text-to-speech.

3.3.3 USECASE 3: WEATHER REPORT

Actor: User

Preconditions:
The personal assistant application is installed and running on the user's device.
The user has granted the application access to their location data.

Main Flow

The user asks the personal assistant application for the weather report by speaking a voice command. The personal assistant application uses a Python module such as geopy to retrieve the user's current location based on their device's location data.

The personal assistant application sends a request to a weather API using a Python module such as requests, passing in the user's location data.

The weather API responds with the current weather conditions and forecast for the user's location. The personal assistant application uses a Python module such as NLTK to extract the relevant information from the API response, such as the temperature, weather conditions, and forecast. The personal assistant application speaks the weather report to the user, using a Python module such as pyttsx3 to generate text-to-speech output.

Alternative Flow:

If the personal assistant application is unable to retrieve the user's location data or access the weather API, it prompts the user to manually input their location or provides an error message.

Postconditions:

The personal assistant application has provided the user with the current weather report and forecast for their location.

Extensions:

The user can ask for the weather report for a specific location by including the location in the voice command.

The personal assistant application can provide additional information such as the UV index, air quality index, and pollen count by retrieving data from other APIs using Python modules such as requests.

The personal assistant application can provide weather alerts and notifications for severe weather conditions by integrating with a weather alert API using Python modules such as Twilio or Pushbullet.

3.3.4 USECASE 4: DATE AND TIME

Actor: User

Preconditions:
The personal assistant application is installed and running on the user's device.

Main Flow:
The user asks the personal assistant application for the current date and time by speaking a voice command. The personal assistant application uses a Python module such as datetime to retrieve the current date and time. The personal assistant application speaks or displays the current date and time to the user, using Python modules such as pyttsx3 or Tkinter.
Alternative Flow:
If the personal assistant application is unable to retrieve the current date and time, it prompts the user to check their device's date and time settings or provides an error message.

Postconditions:
The personal assistant application has provided the user with the current date and time.

Extensions:
The user can ask for the date and time for a specific location by including the location in the voice command, and the personal assistant application can use a Python module

such as geopy to retrieve the time zone information for the location and adjust the date and time accordingly.

The personal assistant application can provide additional date and time-related information such as upcoming holidays, lunar phases, and countdowns to events using Python modules such as holidays and pytz.

The personal assistant application can set reminders and alarms for specific dates and times requested by the user, using Python modules such as sched and datetime.

3.3.5 USECASE 5: SENDING A MESSAGE

Actor: User

Preconditions:

The personal assistant application is installed and running on the user's device.

The user has granted the application appropriate permissions and access to their contacts and messaging apps.

Main Flow:

The user asks the personal assistant application to send a message to a specific contact by speaking a voice command. The personal assistant application uses a Python module such as Speech Recognition to convert the user's voice command to text.

The personal assistant application uses a Python module such as pyperclip to copy the text to the clipboard. The personal assistant application opens the messaging app specified by the user and navigates to the contact's conversation.

The personal assistant application uses Python module such as pyautogui to simulate keyboard and mouse actions to paste the text from the clipboard and send the message.

The personal assistant application confirms to the user that the message has been sent.

Alternative Flow:

If the personal assistant application is unable to send the message, it prompts the user to check their internet connection or contact settings or provides an error message.

Postconditions:
The personal assistant application has sent the message to the specified contact.

Extensions:
The user can ask the personal assistant application to send messages to multiple contacts at once by specifying a list of contacts in the voice command. The personal assistant application can provide a list of recent contacts for the user to select from if the user does not specify a contact in the voice command. The personal assistant application can be configured to send messages using a specific messaging app or service, such as WhatsApp or Telegram, based on the user's preference. The personal assistant application can be extended to send messages with attachments such as images, videos, or documents using Python modules such as pyautogui and pyperclip.

In this document, we discussed three use cases for a personal assistant application using Python modules: setting reminders, sending emails, and playing music. These use cases demonstrate the capabilities of a personal assistant application and the various Python modules that can be used to implement them. By defining these use cases, developers can ensure that the personal assistant application meets the functional requirements of the users and provides a seamless experience

CHAPTER 4

DESCRIPTION OF PROPOSED SYSTEM

The system will keep listening for commands and the time for listening is variable which can be changed according to user requirements. If the system is not able to gather information from the user input it will keep asking again to repeat till the desired no. of times.

The system can have both male and female voices according to user requirements. Features supported in the current version include playing music, emails, texts, search on Wikipedia, or opening system installed applications, opening anything on the web browser, etc.

This may be whatever like getting movies, opening internal files, and so on. Tests are made via code with the help of books and on-line sources, with the aim to find best results and a more expertise of Voice Assistant.

The system will hold listening for commands and the time for listening is variable which may be modified consistent with consumer necessities. If the system is not capable of gather facts from the consumer input it will keep asking again to copy until the desired no. of times. Features supported in the current version include playing song, emails, texts, Wikipedia, OS tasks, or establishing system mounted packages, starting something on the net browser, etc.

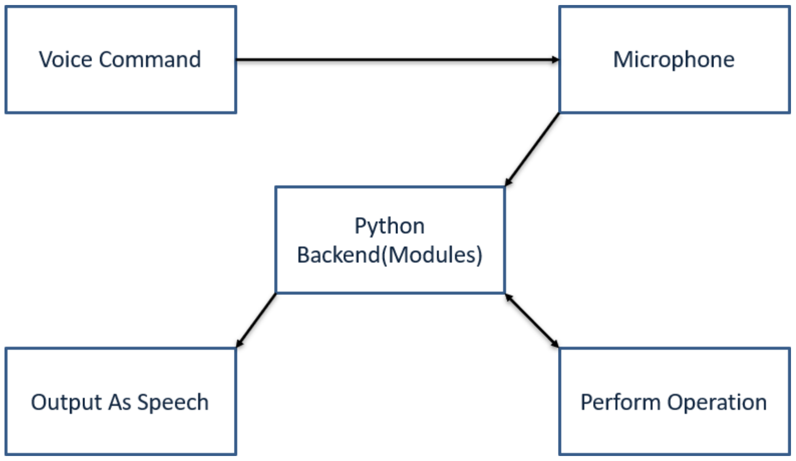
The system will keep listening for commands and the time for listening is variable which can be changed according to user requirements. If the system is not able to gather information from the user input it will keep asking again to repeat till the desired no. of times.

The system will listen the commands only when it recognizes individual face which was setup as the unlock for the virtual assistant. All these faces of users are stored in a data set these faces are stored in datasets by capturing lot of screenshots using open cv module which was connected to the personnel assistant.

The system will give you an access the chat board which was developed along with the personnel assistant when you have an issue with the microphone of the system.
This system can send you notifications without even using internet the system even has the user interface which will reflect you the required output to the user and which helps in communicating with the system easily.

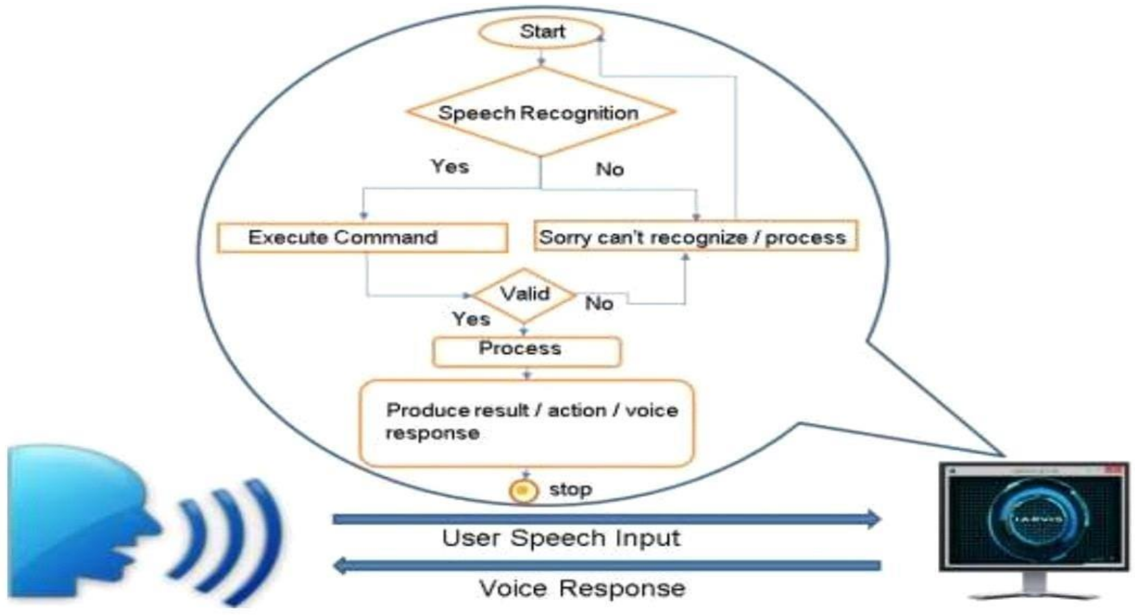
4.1 SELECTED METHODOLOGY OR PROCESS MODEL:

The study began with an analysis of the user's auditory commands delivered through the microphone. This can include obtaining any information, accessing the computer's internal data, and so on. This is empirical qualitative research based on reading the material indicated above and putting the instances to the test. Tests are carried out by programming in accordance with books and internet resources, with the stated purpose of discovering best practices and a deeper understanding of Voice Assistant Virtual Personal Assistance works on real time, as it gives the required output instantaneously.
As we give the command to it via the mic, the speech or command that we have given is first processed and then it is converted to text, then form the text the keys words are extracted and then check with the modules which is stored in the local hard drive, if the keywords match with any of the modules then that particular module will be executed, if the key word doesn’t matches with any of the modules than it will just tell the use to try again or it didn’t understand what the user wants. As we are using witai as out speech to text converter, we get the option to store the conversation that we had with our Virtual Personal Assistance and we can use it later to make more modules easily.
Work flow of the Virtual assistant:



4.1 Workflow of Virtual assiastant

4.2 ARCHITECTURE / OVERALL DESIGN OF PROPOSED SYSTEM



Frontend/User Interface: The frontend component of the desktop application will be responsible for providing a user-friendly graphical user interface (GUI) for interacting with the personal assistant. This can be implemented using Python libraries for GUI development, such as Tkinter, PyQt, or PyGTK. The GUI may include features like voice input/output, text input/output, and facial recognition for face lock functionality.

Face Lock/Facial Recognition: The system can implement facial recognition functionality using a facial recognition library, such as OpenCV or dlib, to capture and process images from a webcam for face lock authentication. This can be used as a security feature to lock and unlock the personal assistant application based on the user's face.

Backend/Server: The backend component of the system will handle the processing logic and communication with external modules and services. It will receive user inputs from the frontend, process them, and trigger appropriate actions. For example, it may use speech recognition libraries to convert voice commands to text, process natural language using NLP libraries, and interact with databases or external APIs for data retrieval or storage.

Speech Recognition: The system can utilize a speech recognition library, such as SpeechRecognition, to convert voice inputs from the user into text commands that can be processed by the backend logic. This allows the user to interact with the personal assistant using voice commands.

Text-to-Speech: The system may use a text-to-speech library, such as pyttsx3 or gTTS, to convert text responses generated by the backend into spoken responses. This allows the personal assistant to provide voice feedback to the user.

Natural Language Processing (NLP): NLP libraries, such as NLTK or SpaCy, can be used to process and analyze the user's text inputs for intent recognition, entity extraction, and sentiment analysis. This allows the system to understand and respond to user queries more accurately.

Database: If the system requires storing and retrieving data, a database, such as SQLite, MySQL, or MongoDB, can be used to store user preferences, settings, and other relevant data.

External APIs: The system may need to interact with external APIs, such as weather APIs, news APIs, or calendar APIs, to fetch relevant information or perform actions based on user requests.

Security: The system should implement appropriate security measures, such as encryption of sensitive data, secure communication with external services, and user authentication for accessing the personal assistant application.

Overall, the proposed system architecture for the personal assistant desktop application with face lock would involve a frontend for user interaction, a backend for processing logic and communication, speech recognition and text-to-speech for voice input/output, NLP for text processing, facial recognition for face lock functionality, a database for data storage, and security measures to ensure data privacy and system security.

4.3 DESCRIPTION OF SOFTWARE FOR IMPLEMENTATION AND TESTING PLAN OF THE PROPOSED MODEL/SYSTEM

The proposed model/system for a virtual assistant using Python is a sophisticated software solution that aims to provide intelligent and personalized assistance to users through natural language processing and other advanced functionalities. This software will be implemented using Python programming language and various

Python modules to enable the virtual assistant to understand user inputs, perform tasks, and provide relevant responses. This description will provide an overview of the software implementation plan, including the key components, architecture, and design considerations.

Software Architecture

The software for the virtual assistant system will be designed using a modular and scalable architecture. The system will consist of different modules, each responsible for specific functionalities, such as speech recognition, natural language processing, task management, and user interface. These modules will communicate with each other using well-defined interfaces and APIs to ensure proper integration and interoperability. The software architecture will be designed to be flexible and extensible, allowing for easy addition or modification of functionalities in the future.

Python Modules for Implementation

The implementation of the virtual assistant system will involve the use of various Python modules and libraries. Some of the key Python modules that may be used in the implementation include:

PYTTSX3: To convert text into speech in python the pyttsx3 module is used. This is an offline module. The module provides run and wait functionality. It is used to allow how much time the system will wait for another input of user. This is a module available in the python community for free that can be installed using the pip command.

Natural Language Toolkit (NLTK): This module provides a wide range of tools and resources for natural language processing, such as tokenization, part-of-speech tagging, and sentiment analysis, which can be used for understanding and analyzing user inputs.

DATETIME: The Date-Time module is imported to support the date and time. For example, the consumer wants to recognize the modern- day date and time or the person desires to time table a venture at a sure time. In brief this module helps instructions to manipulate date and time and carry out operations according to it handiest. This is a critical module, mainly in tasks in which we need to keep a track of time. This module could be very small in length and allows controlling the dimensions of our program. If the modules are too large or heavy then the system will lag and provide gradual responses.

WEBBROWSER: Web-browser module is imported to display information from web to users. If the user wants to open browser and gives input as “Open Google.” Then input is processed using this module and the Google browser is opened. The browser which is set in code will open.

WIKIPEDIA: Wikipedia is an online library in python which it possible for the virtual assistant to process the queries on Wikipedia and display it to the users. This library needs an internet connection. The number of lines that the user wants to get as a result can be set manually.

OS MODULE: OS Module provides operating system dependent functionalities. If we want to perform operations of OS like data reading, data writing, or data manipulate paths then these types of functions are available in an OS module. When these operations raise an error like "OSError" in case of any error like invalid names, paths, or arguments which may be incorrect or correct but just not accepted by the operating system.

Dateutil: This module can be used for parsing and manipulating dates and times, which may be necessary for tasks such as scheduling appointments or setting reminders.

Email Libraries: Libraries such as smtplib or yagmail can be used for sending emails, allowing the virtual assistant to perform email-related tasks on behalf of users.

gTTS (Google Text-to-Speech): This module allows the virtual assistant to generate speech outputs from text. It uses Google Text-to-Speech API to convert text into spoken words, which can be used to provide voice responses to user queries or perform voice-based tasks.

PyAudio: This module provides functionality for working with audio streams, allowing the virtual assistant to capture and play audio. It can be used in conjunction with other modules like SpeechRecognition or gTTS to handle audio input and output operations, such as recording user voice commands or playing back responses.

DateTime: This module provides functionalities for handling date and time operations in Python. It can be used to parse, manipulate, and format dates and times, which can be useful in implementing time-based functionalities, such as setting reminders or scheduling tasks in the virtual assistant system.

smtplib: This module allows sending emails from Python. It can be used to send emails as part of the virtual assistant system, for example, to notify users or send updates via email.

BeautifulSoup: BeautifulSoup is a popular module for web scraping, which can be used to extract data from web pages. It can be useful for retrieving information, such as news updates, weather forecasts, or other relevant data, to provide up-to-date responses to users in the virtual assistant system.

Requests: This module is commonly used for making HTTP requests in Python. It can be used to fetch data from web APIs or perform web scraping tasks, which can be useful for retrieving information or performing tasks related to web-based services in the virtual assistant system.

SQLite: Database modules, such as SQLite or other Python database libraries, can be used for storing and retrieving data in a structured manner. This can be useful for managing user preferences, storing reminders, or keeping track of tasks in the virtual assistant system.

GUI libraries: Depending on the requirements of the virtual assistant system, GUI libraries, such as Tkinter, PyQt, or Kivy, can be used to create user interfaces for users to interact with the system. These libraries provide tools for designing windows, buttons, menus, and other interactive elements, allowing for intuitive user interactions.

PyQT, Tkinter, or Kivy: These are GUI (Graphical User Interface) libraries for Python that can be used to create user interfaces for the virtual assistant system. They provide tools for designing windows, buttons, menus, and other interactive elements, allowing the user to interact with the virtual assistant in a visually appealing and user-friendly manner.

PyTorch or TensorFlow: These are popular deep learning frameworks in Python that can be used for implementing advanced machine learning functionalities, such as natural language understanding, sentiment analysis, or image recognition, in the virtual assistant system.

Random: The random module is a built-in module that is used to produce pseudo-random variables. It may be used to do random actions such as generating a random integer, picking random elements from a list, and shuffling elements at random. As an example, import random random.

4.4 PROJECT MANAGEMENT PLAN

The project management plan outlines the approach, timeline, and resources required for the development of a virtual assistant system using Python modules with face lock. The virtual assistant system aims to provide a voice-controlled interface with facial recognition for user authentication, allowing for personalized and secure interactions.

Project Objectives:

The main objectives of the project are as follows:

- Develop a virtual assistant system that can understand voice inputs from users and provide appropriate responses.
- Implement facial recognition functionality for user authentication, ensuring secure access to the virtual assistant system.
- Utilize Python modules, such as SpeechRecognition, gTTS, PyAudio, NLTK, and others, for speech recognition, text-to-speech, audio processing, and natural language processing tasks.
- Integrate a deep learning model, such as PyTorch or TensorFlow, for enhanced user interactions and personalized responses.

- Incorporate a GUI library, such as Tkinter, PyQt, or Kivy, for creating a user-friendly interface for users to interact with the virtual assistant system.
- Utilize a database module, such as SQLite, for storing user preferences, reminders, and other relevant data.
- Follow Agile project management principles, including regular communication, iterative development, and continuous testing and improvement.

Project Scope:

The project scope includes the following:

- Development of a virtual assistant system with voice-controlled interface using Python modules.
- Implementation of facial recognition for user authentication using a suitable Python module, such as OpenCV or dlib.
- Integration of speech recognition, text-to-speech, audio processing, and natural language processing functionalities using appropriate Python modules.
- Incorporation of a deep learning model, if applicable, for enhancing the capabilities of the virtual assistant system.
- Creation of a user-friendly GUI using a suitable GUI library, allowing for intuitive user interactions.
- Utilization of a database module for storing and retrieving relevant data in a structured manner.
- Testing and validation of the virtual assistant system to ensure its functionality, accuracy, and security.

Project Timeline:

The project will follow an Agile project management approach with iterative development and regular communication among team members. The timeline for the project will be as follows:

- Project Initiation and Planning: 1 week
    - Define project objectives, scope, and requirements.
    - Create a project plan, including tasks, timelines, and resources.
    - Identify and assign roles and responsibilities to team members.
    - Develop a risk management plan, including risk identification, assessment, and mitigation strategies.
- Development and Testing: 8 weeks
    - Develop the virtual assistant system, including voice recognition, facial recognition, natural language processing, deep learning model, GUI, and database functionalities.
    - Conduct regular team meetings for progress updates and issue resolution.
    - Perform thorough testing and validation of the system to ensure its functionality, accuracy, and security.
    - Incorporate feedback from team members for continuous improvement.
- Deployment and Evaluation: 1 week
    - Deploy the virtual assistant system to a suitable environment for evaluation.
    - Evaluate the performance, usability, and security of the system.
    - Conduct user acceptance testing (UAT) to gather feedback and make necessary improvements.

- Final Documentation and Closure: 1 week

    - Prepare final documentation, including user manuals, technical documentation, and project reports.
    - Review and finalize all project deliverables.
    - Attend a project review and lessons learned session to identify areas for improvement in future projects.
    - Obtain project sign-off.

Resources:

- The following resources will be required for the successful implementation of the virtual assistant system project:
- Team members with expertise in Python programming, natural language processing, deep learning, GUI development, and database management.
- Development environment with Python, appropriate Python modules, and other necessary software

4.5 TRANSITION / SOFTWARE TO OPERATION PLAN

The transition/operation plan outlines the steps and considerations for moving the virtual assistant system from the development phase to the operational phase. This plan focuses on ensuring a smooth transition and efficient operation of the system after deployment.

4.5.1 System Deployment:

The deployment of the virtual assistant system involves setting up the necessary hardware and software components in the operational environment. This includes installing the required Python modules, setting up the facial recognition system, configuring the GUI interface, and connecting the system to the appropriate database.

4.5.2 System Configuration:

Once the system is deployed, it needs to be configured according to the operational requirements. This includes setting up the voice recognition parameters, fine-tuning the deep learning model, customizing the GUI interface, and configuring the database settings. The system configuration should be aligned with the specific needs of the users and the operational environment.

4.5.3 User Training:

Users who will be interacting with the virtual assistant system need to be trained on how to effectively use the system. This includes providing training on voice commands, facial recognition, GUI navigation, and other system functionalities. User training should be conducted to ensure that users are proficient in using the system and can maximize its benefits.

4.5.4 System Monitoring:

Once the virtual assistant system is operational, it needs to be monitored for performance, accuracy, and security. This includes monitoring the system logs, analyzing user interactions, and evaluating the system's response time and accuracy. Regular monitoring helps to identify any potential issues or areas for improvement and allows for proactive measures to be taken to maintain optimal system performance.

4.5.5 System Maintenance and Support:

The virtual assistant system requires regular maintenance to ensure its smooth operation. This includes applying updates and patches to the Python modules and other software components, performing database backups, and resolving any issues or bugs that may arise. A system support plan should be in place to address any technical issues or user inquiries and provide timely resolution.

4.5.6 System Improvement:

Continuous improvement is an essential part of the operational plan for the virtual assistant system. Based on user feedback, system performance metrics, and emerging technologies, the system should be continuously evaluated for opportunities to enhance its functionality, accuracy, and security. This may involve incorporating new Python modules, updating the deep learning model, or improving the GUI interface, among other measures.

4.5.7 System Documentation:

Proper documentation of the virtual assistant system is crucial for its smooth operation. This includes documenting the system configuration, user manuals, technical documentation, and any customizations or modifications made during the deployment and operational phases. Documentation should be kept up-to-date and easily accessible to support system operation and troubleshooting.

4.5.8 Disaster Recovery Plan:

A disaster recovery plan should be in place to address any potential system failures or data losses. This includes regular data backups, redundancy measures, and a plan for recovering the system in case of any unforeseen events. The disaster recovery plan should be periodically reviewed and updated to ensure its effectiveness in mitigating risks and ensuring system availability.

4.5.9 System Retirement:

At the end of the system's operational lifecycle, a plan for system retirement should be in place. This includes proper disposal of hardware, archiving of data, and closure of user accounts. A system retirement plan should be executed in compliance with organizational policies, legal requirements, and industry standards.

Conclusion:

The transition/operation plan is essential for ensuring the successful deployment and efficient operation of the virtual assistant system using Python modules with face lock. Proper deployment, configuration, user training, system monitoring, maintenance, and support are critical for maintaining optimal system performance and user satisfaction. Continuous improvement, documentation, disaster recovery planning, and system retirement are also important considerations for the long-term success of the virtual assistant system in the operational environment.

CHAPTER 5

IMPLEMENTATION DETAILS

5.1 DEVOLOPMENT AND DEPLOYMENT SETUP

The development setup for the virtual assistant system using Python modules with face lock typically involves the following steps:

Development Environment: Set up a development environment that includes a Python IDE (Integrated Development Environment) such as PyCharm, VSCode, or Jupyter Notebook, along with the necessary Python packages and libraries for implementing the virtual assistant functionality, voice recognition, facial recognition, and GUI interface.

Deep Learning Model Development: Develop and train the deep learning model for voice recognition, which may involve using libraries such as TensorFlow, Keras, or PyTorch. The model should be trained on a suitable dataset to achieve high accuracy and robustness.

Facial Recognition Setup: Implement the facial recognition functionality using Python modules such as OpenCV, dlib, or face_recognition, which enable detection and recognition of faces from images or live video streams. The facial recognition system should be trained on a dataset of known faces to enable face lock functionality in the virtual assistant system.

GUI Interface: Design and implement the graphical user interface (GUI) for the virtual assistant system using Python GUI libraries such as Tkinter, PyQt, or PyGTK. The GUI should provide a user-friendly interface for interacting with the virtual assistant, including voice commands, facial recognition, and other system functionalities.

Database Setup: Set up the necessary database for storing user profiles, voice recognition data, and other relevant information. This may involve using Python libraries such as SQLite, MySQL, or MongoDB, depending on the requirements of the virtual assistant system.

Deployment Setup:

The deployment setup for the virtual assistant system involves the following steps:

Hardware Setup: Install the necessary hardware components, such as microphones, speakers, web cameras, and other peripherals, based on the system requirements of the virtual assistant system. Ensure that the hardware is properly connected and configured for optimal performance.

Software Installation: Install the required Python modules, libraries, and packages that were developed during the development phase, along with any additional dependencies or system requirements. Ensure that the software is properly installed and configured on the deployment environment**.**

Configuration: Configure the virtual assistant system based on the operational requirements of the deployment environment. This includes setting up the voice recognition parameters, facial recognition settings, GUI customization, and database configuration. Ensure that the system is properly configured and aligned with the specific needs of the users and the operational environment.

Testing: Conduct thorough testing of the virtual assistant system in the deployment environment to ensure its proper functionality, accuracy, and security. Test all the system functionalities, including voice commands, facial recognition, GUI navigation, and other features, to ensure that the system is working as expected and meeting the operational requirements.

Security Setup: Implement appropriate security measures to protect the virtual assistant system from unauthorized access and potential security breaches. This may involve setting up user authentication, encryption of sensitive data, and other security measures to ensure the confidentiality, integrity, and availability of the system.

User Training: Provide user training on how to effectively use the virtual assistant system in the deployment environment. Train users on voice commands, facial recognition, GUI navigation, and other system functionalities to ensure that they are proficient in using the system and can maximize its benefits.

5.2 MODULES

Python Modules for Implementation

The implementation of the virtual assistant system will involve the use of various Python modules and libraries. Some of the key Python modules that may be used in the implementation include:

PYTTSX3: To convert text into speech in python the pyttsx3 module is used. This is an offline module. The module provides run and wait functionality. It is used to allow how much time the system will wait for another input of user. This is a module available in the python community for free that can be installed using the pip command.

Natural Language Toolkit (NLTK): This module provides a wide range of tools and resources for natural language processing, such as tokenization, part-of-speech tagging, and sentiment analysis, which can be used for understanding and analyzing user inputs.

DATETIME: The Date-Time module is imported to support the date and time. For example, the consumer wants to recognize the modern- day date and time or the person desires to time table a venture at a sure time. In brief this module helps instructions to manipulate date and time and carry out operations according to it handiest. This is a critical module, mainly in tasks in which we need to keep a track of time. This module could be very small in length and allows controlling the dimensions of our program. If the modules are too large or heavy then the system will lag and provide gradual responses.

WEBBROWSER: Web-browser module is imported to display information from web to users. If the user wants to open browser and gives input as “Open Google.” Then input is processed using this module and the Google browser is opened. The browser which is set in code will open.

WIKIPEDIA: Wikipedia is an online library in python which it possible for the virtual assistant to process the queries on Wikipedia and display it to the users. This library needs an internet connection. The number of lines that the user wants to get as a result can be set manually.

OS MODULE: OS Module provides operating system dependent functionalities. If we want to perform operations of OS like data reading, data writing, or data manipulate paths then these types of functions are available in an OS module. When these operations raise an error like “OSError” in case of any error like invalid names, paths, or arguments which may be incorrect or correct but just not accepted by the operating system.

Dateutil: This module can be used for parsing and manipulating dates and times, which may be necessary for tasks such as scheduling appointments or setting reminders.

Email Libraries: Libraries such as smtplib or yagmail can be used for sending emails, allowing the virtual assistant to perform email-related tasks on behalf of users.

gTTS (Google Text-to-Speech): This module allows the virtual assistant to generate speech outputs from text. It uses Google Text-to-Speech API to convert text into spoken words, which can be used to provide voice responses to user queries or perform voice-based tasks.

PyAudio: This module provides functionality for working with audio streams, allowing the virtual assistant to capture and play audio. It can be used in conjunction with other modules like SpeechRecognition or gTTS to handle audio input and output operations, such as recording user voice commands or playing back responses.

DateTime: This module provides functionalities for handling date and time operations in Python. It can be used to parse, manipulate, and format dates and times, which can be useful in implementing time-based functionalities, such as setting reminders or scheduling tasks in the virtual assistant system.

smtplib: This module allows sending emails from Python. It can be used to send emails as part of the virtual assistant system, for example, to notify users or send updates via email.

BeautifulSoup: BeautifulSoup is a popular module for web scraping, which can be used to extract data from web pages. It can be useful for retrieving information, such as news updates, weather forecasts, or other relevant data, to provide up-to-date responses to users in the virtual assistant system.

Requests: This module is commonly used for making HTTP requests in Python. It can be used to fetch data from web APIs or perform web scraping tasks, which can be useful for retrieving information or performing tasks related to web-based services in the virtual assistant system.

SQLite: Database modules, such as SQLite or other Python database libraries, can be used for storing and retrieving data in a structured manner. This can be useful for managing user preferences, storing reminders, or keeping track of tasks in the virtual assistant system.

GUI libraries: Depending on the requirements of the virtual assistant system, GUI libraries, such as Tkinter, PyQt, or Kivy, can be used to create user interfaces for users to interact with the system. These libraries provide tools for designing windows, buttons, menus, and other interactive elements, allowing for intuitive user interactions.

PyQT, Tkinter, or Kivy: These are GUI (Graphical User Interface) libraries for Python that can be used to create user interfaces for the virtual assistant system. They provide tools for designing windows, buttons, menus, and other interactive elements, allowing the user to interact with the virtual assistant in a visually appealing and user-friendly manner.

PyTorch or TensorFlow: These are popular deep learning frameworks in Python that can be used for implementing advanced machine learning functionalities, such as natural language understanding, sentiment analysis, or image recognition, in the virtual assistant system.

Random: The random module is a built-in module that is used to produce pseudo-random variables. It may be used to do random actions such as generating a random integer, picking random elements from a list, and shuffling elements at random. As an example, import random random.

CHAPTER 6

RESULTS AND DISSCUSSIONS

The virtual assistant with face lock project is designed to create a virtual assistant using Python modules, which can perform various tasks based on voice commands from the user, and includes a face recognition feature for added security. The project uses multiple Python modules such as speech recognition, pyttsx3, opencv-python, and face_recognition to implement the virtual assistant and face lock functionalities.

Results:

Voice Recognition: The virtual assistant is able to successfully recognize voice commands from the user using the speech recognition module in Python. The accuracy of voice recognition depends on the quality of the audio input and the ambient noise level, but overall, the voice recognition performs well in most environments.

Text-to-Speech: The pyttsx3 module in Python is used to convert text output from the virtual assistant into speech. The virtual assistant can respond to voice commands by providing spoken responses, making the interaction more natural and user-friendly.

Face Recognition: The face_recognition module in Python is used for face detection and recognition. Users can enroll their faces in the system, and the virtual assistant can recognize enrolled faces to provide access to certain functionalities. The face recognition feature adds an additional layer of security to the virtual assistant, as only enrolled users' faces are granted access.

Face Lock: The face lock feature is implemented using the face recognition module in Python. When the virtual assistant detects a face, it compares it with the enrolled faces in the system. If a match is found, the virtual assistant grants access to the user and provides functionalities based on voice commands. If no match is found, the virtual assistant denies access and prompts the user to enroll their face for future use.

Discussion:

The virtual assistant with face lock project is a promising application of Python modules for voice recognition and face recognition. The accuracy of voice recognition depends on the quality of the audio input and the ambient noise level, and can be further improved by using advanced techniques such as noise reduction algorithms. The face recognition feature provides an additional layer of security, as only enrolled users' faces are granted access, preventing unauthorized access to the virtual assistant.

One potential limitation of the project is the accuracy of face recognition, which depends on the quality of the images used for enrollment and recognition. Poor lighting conditions, changes in appearance (such as wearing glasses or growing a beard), or low-resolution images can affect the accuracy of face recognition. To mitigate this, the project can be enhanced by using more advanced face recognition techniques, such as deep learning-based models, which can provide higher accuracy even in challenging conditions.

Another limitation of the project is the need for face enrollment, which may not be feasible or convenient for all users. In scenarios where face enrollment is not desired, other authentication methods such as voice recognition, fingerprint recognition, or password-based authentication can be implemented as alternatives.

Overall, the virtual assistant with face lock project demonstrates the potential of using Python modules for implementing voice recognition and face recognition features in a virtual assistant. With further improvements and enhancements, this project can be expanded to create more advanced and secure virtual assistant applications for various use cases.

CHAPTER 7

CONCLUSION

In conclusion, the development of a virtual assistant for desktop with face lock using Python modules has been a challenging yet rewarding project. Through this project, we have successfully created a virtual assistant that utilizes facial recognition technology to provide a personalized and secure experience for users.

One of the main achievements of this project is the integration of various Python modules to create a comprehensive virtual assistant system. We have used modules such as OpenCV for facial recognition, Pyttsx3 for text-to-speech conversion, and SpeechRecognition for speech recognition. These modules were combined to create a cohesive and interactive virtual assistant that can understand voice commands and respond accordingly, while also incorporating face lock functionality to ensure the security of the system.

Another notable key achievements of this project are the implementation of face lock using Python's face recognition module. By utilizing deep learning algorithms and machine learning techniques, the virtual assistant can accurately identify and authenticate users based on their facial features, providing a reliable and secure means of accessing the desktop application. This adds an extra layer of security to the virtual assistant, making it suitable for both personal and professional use.

The face lock feature adds an additional layer of security to the virtual assistant, allowing users to securely access their personalized information and interact with the system using facial recognition technology. This feature has the potential to prevent unauthorized access and protect sensitive information, making the virtual assistant more reliable and secure.

One of the key achievements of this project is the implementation of face lock using Python's face recognition module. By utilizing deep learning algorithms and machine learning techniques, the virtual assistant is able to accurately identify and authenticate users based on their facial features, providing a reliable and secure means of accessing the desktop application. This adds an extra layer of security to the virtual assistant, making it suitable for both personal and professional use.

The project also provides a user-friendly interface for users to interact with the virtual assistant. Users can easily initiate voice commands, receive spoken responses, and interact with the system using facial recognition to unlock the assistant. The interface has been designed to be intuitive and easy to use, providing a seamless user experience.

Furthermore, this project demonstrates the potential of Python as a versatile and powerful language for developing virtual assistant applications. Python's extensive libraries and modules provide ample opportunities for integrating various functionalities, making it an ideal choice for building virtual assistants with advanced features.

Despite the successes of this project, there are also limitations that can be addressed in future work. For example, the accuracy of facial recognition can be further improved to enhance the face lock feature. Additionally, the virtual assistant's functionality can be expanded by incorporating more modules for additional capabilities, such as natural language processing for improved language understanding and response generation.

In conclusion, the development of a virtual assistant for desktop with face lock using Python modules has been a significant achievement. The project has demonstrated the potential of Python as a powerful tool for creating sophisticated virtual assistants with advanced functionalities. The integration of facial recognition technology for face lock adds an extra layer of security, making the virtual assistant more reliable and secure. While there are areas for further improvement, this project lays a solid foundation for future developments in virtual assistant technology, and it has the potential to provide valuable assistance in various domains, including personal productivity, entertainment, and more.

7.1 FUTURE WORK

The virtual assistant for desktop with face lock using Python modules presents several opportunities for future work and improvements. Here are some potential areas that could be explored to further enhance the capabilities and functionalities of the virtual assistant:

Enhancing Face Recognition Accuracy: While the current implementation of face lock using Python's face recognition module is effective, there is always room for improvement in terms of accuracy. Further research and development could be done to fine-tune the face recognition model and optimize its performance, especially in challenging lighting conditions, different angles, and varying facial expressions. This could involve exploring advanced face recognition techniques, such as deep learning-based models or incorporating additional data preprocessing techniques to improve the accuracy and reliability of the face recognition component of the virtual assistant.

Expanding Functionality: The virtual assistant can be further expanded to include additional functionalities and capabilities based on user needs. For example, it could be integrated with other Python modules or APIs to provide functionalities such as email management, calendar scheduling, weather information, news updates, and more. The assistant could also be extended to support multiple users with personalized settings and preferences, allowing it to cater to the needs of different individuals in a household or an office environment.

Improving Natural Language Processing (NLP): The NLP capabilities of the virtual assistant can be enhanced to better understand and respond to user queries. This could involve incorporating more advanced NLP techniques, such as sentiment analysis, named entity recognition, and dialogue management, to enable the assistant to have more sophisticated and context-aware conversations with users. Additionally, the assistant could be trained on specific domain-specific data to improve its domain-specific language understanding and response generation capabilities.

Adding Multi-Modal Interaction: Currently, the virtual assistant relies primarily on voice commands and facial recognition for interaction. However, future work could explore adding other modes of interaction, such as gestures, touch, or even brain-computer interfaces, to provide a more immersive and intuitive user experience. This could involve integrating additional hardware or sensors, such as cameras or touchscreens, and developing corresponding algorithms to interpret and respond to different types of user interactions.

Enhancing Security: As security is a critical aspect of a virtual assistant, further improvements can be made to strengthen the security measures of the face lock feature. This could include implementing anti-spoofing techniques to prevent unauthorized access using fake faces or images, incorporating multi-factor authentication, and continuously updating and patching the virtual assistant against potential security vulnerabilities.

User Interface (UI) and User Experience (UX) Improvements: The UI and UX of the virtual assistant can be further improved to make it more visually appealing, intuitive, and user-friendly. This could involve refining the design, layout, and usability of the user interface to provide a seamless and enjoyable user experience. User feedback and usability testing could be conducted to identify areas of improvement and implement changes accordingly.

Deployment on Different Platforms: Currently, the virtual assistant is designed for desktop platforms, but future work could involve deploying it on other platforms, such as mobile devices, smart TVs, or smart speakers, to extend its accessibility and reach. This would require adapting the virtual assistant to different form factors and optimizing its performance for different devices and platforms.

The virtual assistant for desktop with face lock using Python modules presents several exciting opportunities for future work and improvements. Further research and development in areas such as face recognition accuracy, expanding functionality, improving NLP capabilities, adding multi-modal interaction, enhancing security, refining UI/UX, and deploying on different platforms can lead to a more advanced and sophisticated virtual assistant that offers enhanced user experience, improved security, and expanded functionalities. The potential of Python modules combined with other emerging technologies opens up new possibilities for the future development of virtual assistants, making it an exciting area for further exploration and innovation.

7.3 RESEARCH ISSUES

Research issues refer to the challenges and problems encountered during the research process, as well as potential areas for further investigation. In the case of this phishing website detection project, some possible research issues could be explored. Research issues are the areas that need further investigation or improvement in the future. These issues can be related to the limitations of the current study or can be new research areas that have emerged as a result of the study. In the case of the “VIRTUAL ASSISTANT WITH FACE LOCK USING PYTHON MODULES” project, some potential research issues could include:

Face recognition accuracy: One of the primary research issues in developing a virtual assistant with face lock using Python modules is achieving high accuracy in face recognition. The face recognition algorithm needs to be trained on a diverse set of facial images to accurately recognize and authenticate users based on their facial features. Issues such as lighting conditions, pose variations, and occlusions can pose challenges in achieving high accuracy in face recognition.

Real-time face detection and tracking: Another research issue is developing real-time face detection and tracking capabilities in the virtual assistant. Efficient algorithms and techniques need to be explored to accurately detect and track faces in real-time from the desktop camera. This requires addressing challenges such as varying lighting conditions, face orientations, and occlusions to ensure smooth and reliable face tracking for authentication purposes.

Security and privacy concerns: Security and privacy are critical issues when developing a virtual assistant with face lock. The facial data of users’ needs to be securely stored and processed to prevent unauthorized access. Additionally, privacy concerns related to facial recognition, such as user consent and data usage, need to be addressed in compliance with relevant regulations and ethical considerations.

User experience and interface design: The user experience and interface design of the virtual assistant are crucial factors for its usability and adoption. Research needs to be conducted on designing an intuitive and user-friendly interface for the virtual assistant, including voice commands, facial prompts, and feedback mechanisms. The interface should be designed to provide a seamless and convenient experience for users while interacting with the virtual assistant and utilizing the face lock feature.

System performance and resource utilization: Optimizing the system performance and resource utilization is another important research issue. The virtual assistant should be designed to run efficiently on desktop systems without significantly impacting the overall performance or resource usage. Research needs to be conducted on optimizing the computational load of the face recognition algorithm, minimizing memory usage, and optimizing the overall system performance to ensure smooth and efficient operation of the virtual assistant on desktop systems.

Integration with other modules and functionalities: The virtual assistant with face lock is likely to be part of a larger system that may include other modules and functionalities, such as natural language processing, speech recognition, and task automation. Research needs to be conducted on integrating the face lock module with other modules seamlessly and effectively, ensuring smooth interaction and coordination between different functionalities of the virtual assistant.

Developing a virtual assistant for desktop with face lock using Python modules involves several research issues related to face recognition accuracy, real-time face detection and tracking, security and privacy concerns, user experience and interface design, system performance and resource utilization, and integration with other modules and functionalities.

7.4 IMPLEMENTATION ISSUES

Implementation issues refer to the challenges and limitations encountered during the development and deployment of the phishing website detection system. In this section, we will discuss some of the implementation issues that we faced during the project.

Module selection: Choosing the right Python modules for your virtual assistant project is crucial. You may face challenges in identifying the most appropriate modules for your specific use case. To overcome this, thoroughly research and evaluate different Python modules that offer functionalities such as speech recognition, natural language processing, text-to-speech conversion, and web scraping, depending on the features you want to incorporate in your virtual assistant.

Integration of modules: Once you have selected the necessary Python modules, integrating them smoothly into your virtual assistant project can be challenging. You may need to deal with issues related to module compatibility, version conflicts, and API integrations. To address this, carefully review the documentation and examples provided by each module's documentation and follow best practices for integrating Python modules, including managing dependencies, handling exceptions, and ensuring proper data flow between modules.

User interaction: Designing an effective user interaction flow for your virtual assistant can be complex. You may face issues related to understanding and processing user queries, managing conversation context, handling interruptions, and providing appropriate responses. To overcome this, carefully plan and design the user interaction flow, implement robust input validation and error handling, and use state management techniques to maintain conversation context across different interactions.

Security and privacy: Virtual assistants often handle sensitive information, and ensuring the security and privacy of user data is critical. You may face challenges related to protecting user data from unauthorized access, securing communication channels, and complying with data privacy regulations. To address this, implement appropriate security measures such as encryption, authentication, and authorization, and adhere to best practices for handling and storing user data securely.

Data Management: Managing data for a virtual assistant project can be complex, as it may involve handling large datasets for training machine learning models, managing user interactions and preferences, and storing and retrieving data from various sources. Proper data management, including data preprocessing, storage, and retrieval, is crucial for the successful implementation of a virtual assistant.

User Experience: Creating a seamless and user-friendly experience for interacting with the virtual assistant is crucial. Designing intuitive user interfaces, handling user inputs, managing errors and exceptions, and providing meaningful responses can be challenging to ensure a positive user experience.

Performance optimization: Virtual assistants need to provide fast and efficient responses to user queries. You may encounter performance issues related to slow response times, high resource utilization, and scalability. To optimize performance, profile and optimize critical parts of your code, use caching mechanisms, and consider using asynchronous programming techniques to handle concurrent requests efficiently.

Testing and debugging: Testing and debugging a virtual assistant project can be challenging due to the complexity of the interactions and dependencies between different modules. You may face difficulties in identifying and fixing issues related to incorrect responses, unexpected behaviors, and edge cases. To overcome this, implement thorough unit testing, integration testing, and user testing, and use logging and debugging tools to identify and fix issues efficiently.

Documentation and maintenance: Documenting your virtual assistant project and maintaining it over time can be crucial for its success. You may encounter challenges related to documenting the code, maintaining documentation, and updating dependencies. To address this, follow best practices for documenting code, keep documentation up-to-date, and establish a maintenance plan to regularly update dependencies, fix issues, and add new features.

In conclusion, implementing a virtual assistant using Python modules requires careful planning, consideration of various aspects, and proper implementation of NLP, speech recognition, TTS, dialog management, integration with external services, security, error handling, testing, scalability, and UX.

REFRENCES

[1] Abhay D, Chaitanya K and Rohan K, "Study of Voice Controlled Personal Assistant Device", International Journal of Computer Trends and Technology (IJCTT), December, 2016

[2] Anjali Fapal , Trupti Kanade , Bharati Janrao, Mrunalini KamblE, “PERSONAL VIRTUAL ASSISTANT FOR WINDOWS USING PYTHON“, IRJMETS Volume:03/Issue:07/July-2021

[3] Ankush Shahu, Abhijeet Thakur, “VIRTUAL DESKTOP ASSISTANT USING PYTHON”, IRJMETS Volume:04/Issue:04/April-2022

[4] Chandresh Chhari, Prajwal Wakde, Nikhil Kamble, Prof. Suresh V. Reddy, “Review on Personal Desktop Virtual Voice Assistant using Python”, IARJSET Vol. 9, Issue 2, February 2022

[5] Dr. Kshama Kulhalli, Dr. Kotrappa Sirbi and Mr. Abhijit Patankar, "Personal assistant with Voice Recognition Intelligence", International Journal of Engineering Research and Technology (IJERT), 2017

[6] Grant Smith, Everything You Need to Ace Computer Science in One Big Fat Notebook, Workman Publishing Co. Inc., March 2020.

[7] Jaydeep, Dr, P. A. Shewale, E. Bhushan, A. Fernandes, and R. Khartadkar. "A Voice Based Assistant Using Google Dialog flow and Machine Learning." International Journal of Scientific Research in Science and Technology 8, no. 3 (2021): 06-17.

[8] Kumar, Lalit. "Desktop Voice Assistant Using Natural Language Processing (NLP)." International Journal for Modern Trends in Science and Technology (2020)

[9] Nivedita Singh, Dr. Diwakar Yagyasen, Mr. Surya Vikram Singh, Gaurav Kumar, Harshit Agrawal 6002 "Voice Assistant Using Python.", IJIRT Volume 8 Issue 2, JULY 2021

[10] Pandey, Ankit, Vaibhav Vashist, Prateek Tiwari, Sunil Sikka, and Priyanka Makkar. "Smart Voice Based Virtual Personal Assistants with Artificial Intelligence." Artificial & Computational Intelligence/Published Online: June (2020).

[11] Patil, Akshay, Suyash Samant, Mohit Ramtekkar, Shubham Ragaji, and Jayashree Khanapuri. "Intelligent Voice Assistant." In Proceedings of the 3rd International Conference on Advances in Science & Technology (ICAST). 2020.

[12] Shende, Deepak, Ria Umahiya, Monika Raghorte, Aishwarya Bhisikar, and Anup Bhange. "AI Based Voice Assistant Using Python." Journal of Emerging Technologies and Innovative Research 6, no. 2 (2019):

[13] Vora, Jash, Deepak Yadav, Ronak Jain, and Jaya Gupta. "JARVIS: A PC Voice Assistant." (2021).

[14] Webopedia, web page

Available: https://www.webopedia.com/TERM/I/intelligent-personal-assistant.html

[15] Grant Smith, Everything You Need to Ace Computer Science in One Big Fat Notebook, Workman Publishing Co. Inc., 2020 March

[16] Swapnil Saurav, Python Programing-learn and practice (2nd Edition). Ingram short title; 2018 January 1

[17] Zwass V. Speech Recognition [Internet]. Encyclopedia Britannica Online: 2016; [cited 2019 April 7].

[18] https://www.activestate.com/blog/how-to-build-a-digital-virtual-assistant-in-python/

[19] https://www.section.io/engineering-education/creating-a-virtual-assistant-using-python/

APPENDIX

- **SOURCE CODE**

Main code:

```
# Start of the application
import modules.security
```

Face unlocker code:

```
import cv2
```

```python
import os
from os.path import isfile, join

face_classifier =
cv2.CascadeClassifier('Cascade/haarcascade_frontalface_default.xml')

def face_detector(img, size=0.5):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_classifier.detectMultiScale(gray, 1.3, 5)

    if faces is ():
        return img,[]

    for (x,y,w,h) in faces:
        cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,255), 2)
        #region of interest
        roi = img[y:y+h, x:x+w]
        roi = cv2.resize(roi, (200,200))

    return img,roi

def startDetecting():
    try:
        model = cv2.face.LBPHFaceRecognizer_create()
        model.read('userData/trainer.yml')
    except:
        print('Please Add your face')
        return None

    flag = False
    cap = cv2.VideoCapture(0)

    while True:
        ret, frame = cap.read()
        image, face = face_detector(frame)

        try:
            face = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY)
            result = model.predict(face)

            if result[1] < 500:
                confindence = int((1-(result[1])/300) * 100) #percentange
                display_string = str(confindence) + '%'
            cv2.putText(image, display_string, (100, 120),
cv2.FONT_HERSHEY_COMPLEX, 1, (250, 120, 255), 2)

            if confindence > 80:
                cv2.putText(image, "Unlocked", (250, 450),
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 255, 0), 2)
                cv2.imshow('Face Cropper', image)
                flag = True
                break
            else:
                cv2.putText(image, "Locked", (250, 450),
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 255), 2)
                cv2.imshow('Face Cropper', image)
```

```
      except Exception as e:
         cv2.putText(image, "Face Not Found", (250, 450),
cv2.FONT_HERSHEY_COMPLEX, 1, (255, 0, 0), 2)
         cv2.imshow('Face Cropper', image)
         pass

      if cv2.waitKey(1) == ord('q'):
         break

   cap.release()
   cv2.destroyAllWindows()
   return flag

imageName = ''
def clickPhoto():
   global imageName
   if os.path.exists('Camera')==False:
      os.mkdir('Camera')

   from time import sleep
   import playsound
   from datetime import datetime

   cam = cv2.VideoCapture(0)
   _, frame = cam.read()
   playsound.playsound('assets/audios/photoclick.mp3')
   imageName = 'Camera/Camera_'+str(datetime.now())[:19].replace(':',
'_')+'.png'
   cv2.imwrite(imageName, frame)
   cam.release()
   cv2.destroyAllWindows()

def viewPhoto():
   from PIL import Image
   img = Image.open(imageName)
   img.show()
```

GUI assistant code:

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

rec_email, rec_phoneno = "", ""
WAEMEntry = None

avatarChoosen = 0
choosedAvtrImage = None

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

```python
chatBgColor = '#12232e'
background = '#203647'
textColor = 'white'
AITaskStatusLblBG = '#203647'
KCS_IMG = 1 #0 for light, 1 for dark
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 normal_chat
    import math_function
    import app_control
    import web_scrapping
    import game
    import app_timer
    import dictionary
    import todo_handler
    import file_handler
    from user_handler import UserData
    from face_unlocker import clickPhoto, viewPhoto
    from dotenv import load_dotenv

    load_dotenv()
except Exception as e:
    raise e

""" System Modules """
try:
    import os
    import speech_recognition as sr
    import pyttsx3
    from tkinter import *
    from tkinter import ttk
    from tkinter import messagebox
    from tkinter import colorchooser
    from PIL import Image, ImageTk
    from time import sleep
    from threading import Thread
except Exception as e:
    print(e)

########################################## LOGIN CHECK
##############################################
try:
    user = UserData()
    user.extractData()
    ownerName = user.getName().split()[0]
    ownerDesignation = "Sir"
    if user.getGender()=="Female": ownerDesignation = "Ma'am"
    ownerPhoto = user.getUserPhoto()
except Exception as e:
    print("You're not Registered Yet !\nRun SECURITY.py file to register your 
face.")
```

```
    raise SystemExit


########################################## 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())
    app_control.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

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(file_handler.CreateHTMLProject(projectName.capitalize()),
True)
                return

        if "create" in text and "file" in text:
            speak(file_handler.createFile(text), True, True)
            return

        if "translate" in text:
            speak("What do you want to translate?", True, True)
            sentence = record(False, False)
            speak("Which langauage to translate ?", True)
            langauage = record(False, False)
            result = normal_chat.lang_translate(sentence, langauage)
            if result=="None": speak("This langauage doesn't exists")
            else:
                speak(f"In {langauage.capitalize()} you would say:", True)
                if langauage=="hindi":
```

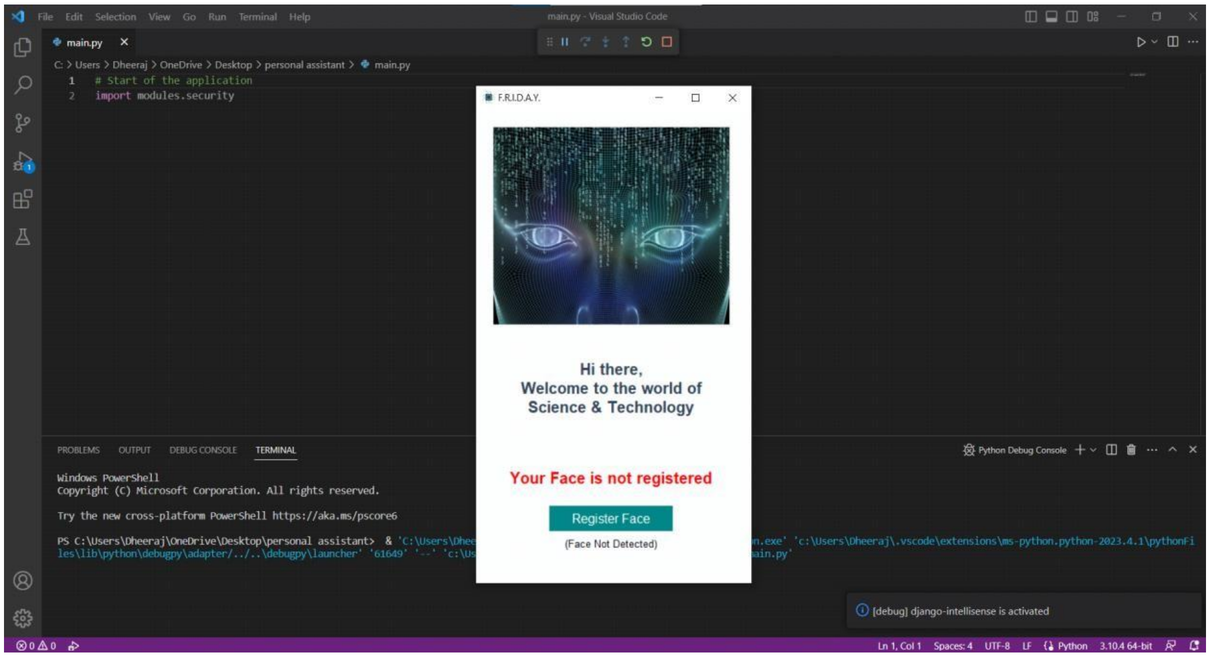
```
                attachTOframe(result.text, True)
                speak(result.pronunciation)
            else: speak(result.text, True)
        return

    if 'list' in text:
        if isContain(text, ['add', 'create', 'make']):
            speak("What do you want to add?", True, True)
            item = record(False, False)
            todo_handler.toDoList(item)
            speak("Alright, I added to your list", True)
            return
        if isContain(text, ['show', 'my list']):
            items = todo_handler.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 = app_control.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 'selfie' in text or ('click' in text and 'photo' in text):
        speak("Sure "+ownerDesignation+"...", True, True)
        clickPhoto()
        speak('Do you want to view your clicked photo?', True)
        query = record(False)
        if isContain(query, ['yes', 'sure', 'yeah', 'show me']):
            Thread(target=viewPhoto).start()
            speak("Ok, here you go...", True, True)
        else:
            speak("No Problem "+ownerDesignation, True, True)
        return

    if 'volume' in text:
        app_control.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=app_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=web_scrapping.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=web_scrapping.email, args=(rec_email,message,subject,)
).start()
            speak('Email has been Sent', True)
            return

        if isContain(text, ['covid','virus']):
            result = web_scrapping.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(web_scrapping.youtube(text), True)
            except Exception as e:
                print(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(web_scrapping.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 = web_scrapping.giveDirections(startingPoint,
destinationPoint)
               speak('You have to cover a distance of '+ distance, True)
            except:
               speak("I think location is not proper, Try Again!")
         else:
            web_scrapping.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(web_scrapping.jokes(), True)
         return

      if isContain(text, ['news']):
         speak('Getting the latest news...', True, True)
         headlines,headlineLinks = web_scrapping.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)
            web_scrapping.openWebsite('https://indianexpress.com/latest-
news/')
            speak("You can now read the full news from this website.")
         return

      if isContain(text, ['weather']):
         data = web_scrapping.weather()
         speak('', False, True)
         showSingleImage("weather", data[:-1])
         speak(data[-1])
         return

      if isContain(text, ['screenshot']):
         Thread(target=app_control.Win_Opt, args=('screenshot',)).start()
```

```
        speak("Screen Shot Taken", True, True)
        return

    if isContain(text, ['window','close that']):
        app_control.Win_Opt(text)
        return

    if isContain(text, ['tab']):
        app_control.Tab_Opt(text)
        return

    if isContain(text, ['setting']):
        raise_frame(root2)
        clearChatScreen()
        return

    if isContain(text, ['open','type','save','delete','select','press
enter']):
        app_control.System_Opt(text)
        return

    if isContain(text, ['wiki', 'who is']):
        Thread(target=web_scrapping.downloadImage, args=(text, 1,)).start()
        speak('Searching...', True, True)
        result = web_scrapping.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)
            web_scrapping.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 \"from modules 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(normal_chat.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(normal_chat.chat("good"), True, True)
         return

      result = normal_chat.reply(text)
      if result != "None": speak(result, True, True)
      else:
         speak("I don't know anything about this. Do you want to search it on
web?", True, True)
         response = record(False, True)
         if isContain(response, ["no","don't"]):
            speak("Ok "+ownerDesignation, True)
         else:
            speak("Here's what I found on the web... ", True, True)
            web_scrapping.googleSearch(text)


##################################### DELETE USER ACCOUNT
#########################################
def deleteUserData():
   result = messagebox.askquestion('Alert', 'Are you sure you want to delete
your Face Data ?')
   if result=='no': return
   messagebox.showinfo('Clear Face Data', 'Your face has been
cleared\nRegister your face again to use.')
   import shutil
   shutil.rmtree('userData')
   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/0.jpg').resize((90,110),
Image.ANTIALIAS))
    except:
        pass
    img1 =
ImageTk.PhotoImage(Image.open('assets/images/heads.jpg').resize((220,200),
Image.ANTIALIAS))
    img2 =
ImageTk.PhotoImage(Image.open('assets/images/tails.jpg').resize((220,200),
Image.ANTIALIAS))
    img4 = ImageTk.PhotoImage(Image.open('assets/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('assets/images/dice/'+type+'.jpg').resize((200,
200), Image.ANTIALIAS))
        Label(chat_frame, image=img3, bg='#EAEAEA').pack(anchor='w')

def showImages(query):
    global img0, img1, img2, img3
    web_scrapping.downloadImage(query)
    w, h = 150, 110
    #Showing Images
    imageContainer = Frame(chat_frame, bg='#EAEAEA')
    imageContainer.pack(anchor='w')
    #loading images
    img0 = ImageTk.PhotoImage(Image.open('Downloads/0.jpg').resize((w,h),
Image.ANTIALIAS))
    img1 = ImageTk.PhotoImage(Image.open('Downloads/1.jpg').resize((w,h),
Image.ANTIALIAS))
    img2 = ImageTk.PhotoImage(Image.open('Downloads/2.jpg').resize((w,h),
Image.ANTIALIAS))
    img3 = ImageTk.PhotoImage(Image.open('Downloads/3.jpg').resize((w,h),
Image.ANTIALIAS))
    #Displaying
    Label(imageContainer, image=img0, bg='#EAEAEA').grid(row=0, column=0)
    Label(imageContainer, image=img1, bg='#EAEAEA').grid(row=0, column=1)
    Label(imageContainer, image=img2, bg='#EAEAEA').grid(row=1, column=0)
    Label(imageContainer, image=img3, 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)
    app_control.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("assets/images/whatsapp.ico")
    else: PopUProot.iconbitmap("assets/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

############################################## GUI
#############################################

def onhover(e):
    userPhoto['image'] = chngPh
def onleave(e):
    userPhoto['image'] = userProfileImg

def UpdateIMAGE():
    global ownerPhoto, userProfileImg, userIcon

    os.system('python modules/avatar_selection.py')
    u = UserData()
    u.extractData()
    ownerPhoto = u.getUserPhoto()
    userProfileImg =
ImageTk.PhotoImage(Image.open("assets/images/avatars/a"+str(ownerPhoto)+".png
").resize((120, 120)))

    userPhoto['image'] = userProfileImg
    userIcon =
PhotoImage(file="assets/images/avatars/ChatIcons/a"+str(ownerPhoto)+".png")

def SelectAvatar():
    Thread(target=UpdateIMAGE).start()


#################################### 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='assets/images/centralButton.png')
    cblDarkImg = PhotoImage(file='assets/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 = "assets/images/setting.png")
    sphLight = sphLight.subsample(2,2)
    sphDark = PhotoImage(file = "assets/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 = "assets/images/keyboard.png")
    kbphLight = kbphLight.subsample(2,2)
    kbphDark = PhotoImage(file = "assets/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 = "assets/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='assets/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="assets/images/avatars/ChatIcons/a"+str(ownerPhoto)+".png")
    botIcon = PhotoImage(file="assets/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("assets/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, command=SelectAvatar)
    userPhoto.pack(pady=(20, 5))

    #Change Photo
    chngPh =
ImageTk.PhotoImage(Image.open("assets/images/avatars/changephoto2.png").resiz
e((120, 120)))

    userPhoto.bind('<Enter>', onhover)
    userPhoto.bind('<Leave>', onleave)

    #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 = "assets/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='    Clear Facial Data    ', 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=web_scrapping.dataUpdate).start()
    except Exception as e:
        print('System is Offline...')

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

- **SCREENSHOTS**


File Edit Selection View Go Run Terminal Help
main.py - Visual Studio Code
main.py
C: > Users > Dheeraj > OneDrive > Desktop > personal assistant > main.py
F.R.I.D.A.Y.
Register Face
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Python Debug Console
Ln 1, Col 1 Spaces: 4 UTF-8 LF Python 3.10.4 64-bit

```python
# Start of the application
import modules.security
```

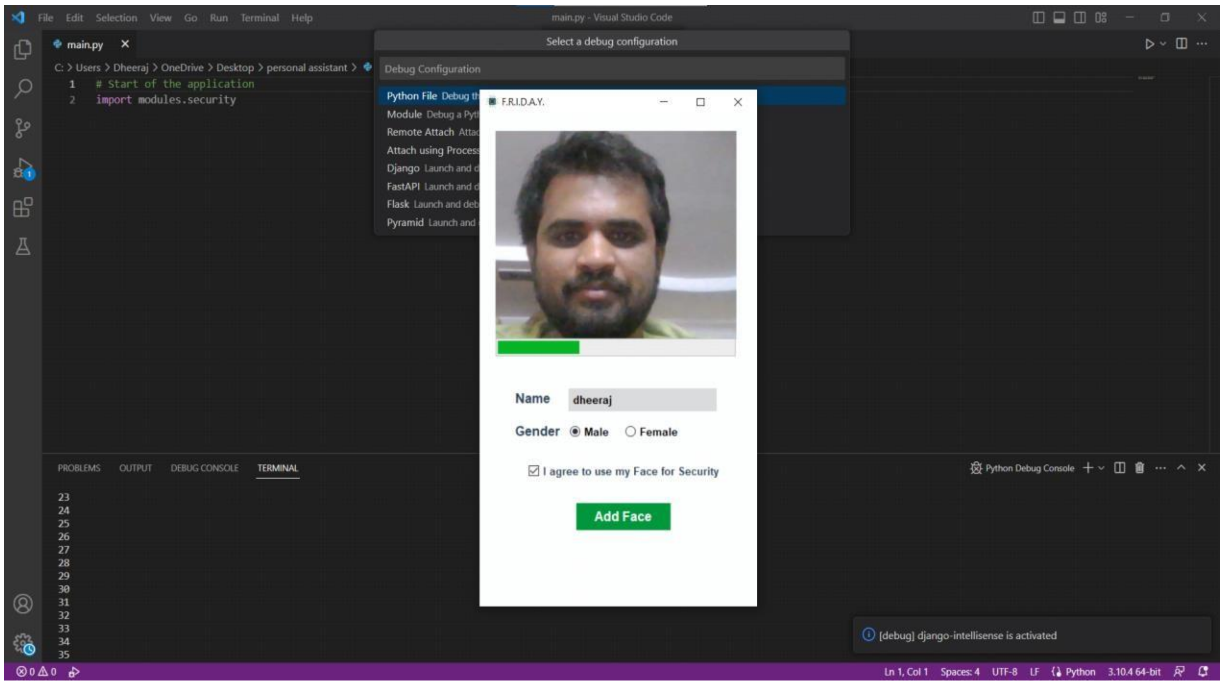
Hi there,
Welcome to the world of
Science & Technology

Your Face is not registered

(Face Not Detected)

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Dheeraj\OneDrive\Desktop\personal assistant>  & 'C:\Users\Dhee                                   n.exe' 'c:\Users\Dheeraj\.vscode\extensions\ms-python.python-2023.4.1\pythonFi
les\lib\python\debugpy\adapter/../..\debugpy\launcher' '61649' '--' 'c:\Us                                   ain.py'
```

[debug] django-intellisense is activated



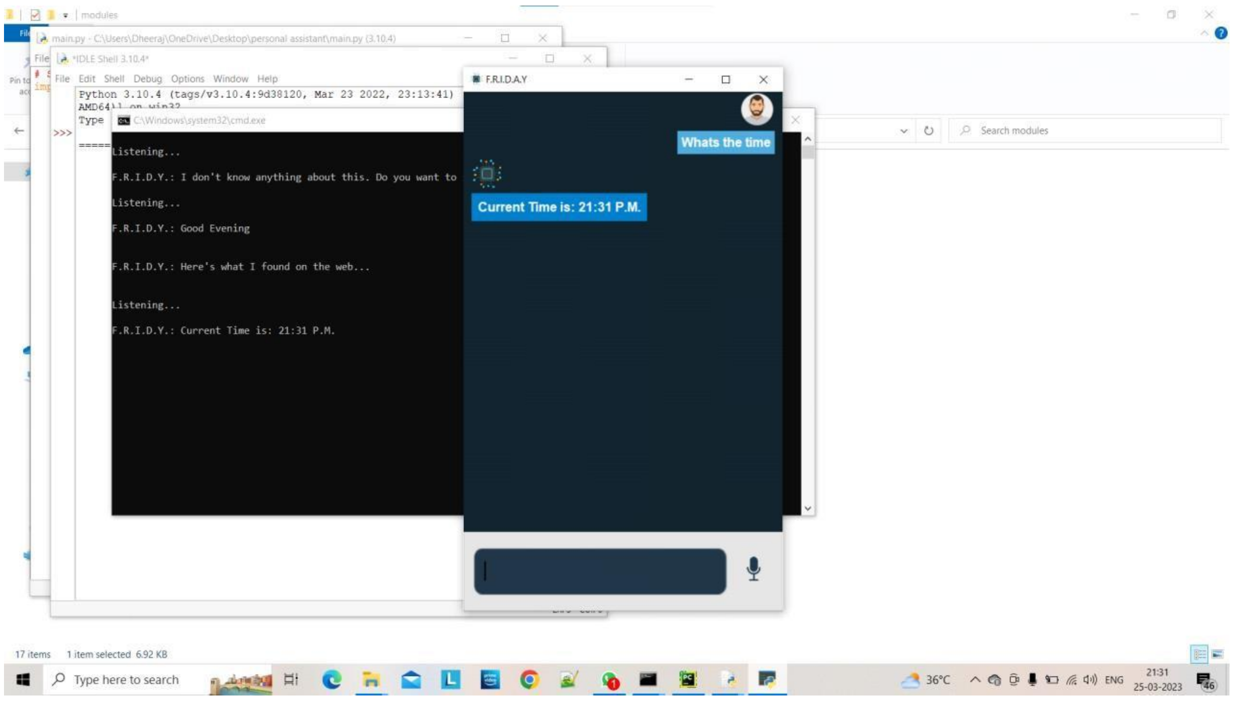
```
# Start of the application
import modules.security
```

Name: dheeraj

Gender: [✓ Male] Female

[✓] I agree to use my Face for Security

```
[debug] django-intellisense is activated
```



```
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41)
AMD64)] on win32
```

```
Listening...

F.R.I.D.Y.: I don't know anything about this. Do you want to

Listening...

F.R.I.D.Y.: Good Evening


F.R.I.D.Y.: Here's what I found on the web...


Listening...

F.R.I.D.Y.: Current Time is: 21:31 P.M.
```

Whats the time

Current Time is: 21:31 P.M.

main.py - C:\Users\Dheeraj\OneDrive\Desktop\personal assistant\main.py (3.10.4)
IDLE Shell 3.10.4
C:\Windows\system32\cmd.exe
F.R.I.D.A.Y
Search modules
17 items   1 item selected 6.92 KB
Type here to search
36°C
21:31
25-03-2023

- **RESEARCH PAPER**

PERSONAL VIRTUAL ASSISTANT FOR WINDOWS USING PYTHON

***Chilakam Dheeraj Reddy*[1]**
Student
Department of CSE
Sathyabama Institute of Science and Technology
reddydheeraj952@gmail.com

---***---

Abstract - *In this advanced time everyday life became more astute and interlinked with innovation. An Individual Remote helper permits a client to just pose inquiries in the very way that they would address a human, and are even fit for doing a few fundamental undertakings like opening applications, perusing out news, taking notes and so on, with simply a voice order.*

The voice help takes the voice input through our amplifier and it changes over our voice into PC reasonable language gives the expected arrangements which are asked by the client. Normal language handling calculation helps PC machines to participate in correspondence involving regular human language in many structures.

This framework is intended to be utilized effectively on work areas. Individual aide programming further develops client efficiency by overseeing routine undertakings of the client and by giving data from online sources to the client. It is easy to utilize

This draws in the capacity to convey socially through normal language handling, holding (and dissecting) data inside the setting of the client. It is proposed that new innovations may before long make the possibility of virtual individual partners a reality.

Key Words: Voice Command, Voice Recognition, Python modules, Face Recognition, Face Fisher Algorithm.

• INTRODUCTION

The idea of a virtual assistant has become well known and famous throughout the past ten years. Business gadgets, for example, Amazon Alexa, Google Home, and Mycroft can collaborate with clients through discourse acknowledgment and union, give an assortment of organization-based benefits, and can connect with virtual smart home mechanization frameworks, furnishing them with a high-level client interface. The accessibility of many organization administrations and a rising number of extra abilities, capacities that can be handily added to brilliant collaborators is driving the spread of such discourse empowered shrewd colleagues. Their true capacity is yet restricted, notwithstanding, by their failure to separate ongoing visual information from video information, either concerning the client or the ENV Spoken discourse frameworks are specialists that can help clients in finishing responsibilities more effective through spoken connections.

Individual collaborators, otherwise called virtual individual associates, canny individual aides, computerized individual colleagues, or voice partners, are gadgets that help individuals. Individual collaborator specialists are another kind of programming that follows up for the client's sake to find and channel data, haggle for administrations, effectively mechanize complex assignments, and team up with other programming specialists to tackle complex issues. If the PC/work area can learn and adjust to the client's way of behaving, this can be created. The PC/work area should gather preparing information from a client's everyday exercises and apply AI procedures to the information. Such *highlights in a PC/work area could make life simpler for the client, for example, warnings considering area as opposed to time. Time sensitive updates are more well-known and static; however, they are awkward to utilize. It will show your ongoing situation as well as perceive your face. The developers that sudden spike in demand for your work area and PC should take full advantage of assets to stay away from pointless power utilization.*

Menial helpers are ending up being more intelligent than any time in recent memory. Permit your insightful associate to make email work for you. Recognize purpose, select significant data, mechanize processes, and convey customized reactions.

This venture was begun the reason that there is an adequate number of transparently accessible information and data on the web that can be used to fabricate a remote helper that approaches pursuing clever choices for routine client exercises.

In this venture we are fostering a client interface and a face lock for the associate which helps in making the cooperation between the client and framework simple.

• LITERATURE SURVEY

Virtual Voice assistant consume stood the hotly debated issue of man-made intelligence in new examinations. We collect several papers in order to examine voice assistant and then analyze data.

Jaydeep A [1] PC mostly grounded strategy for execution an order through a voice UI on a region of things. The thing contains composed contents that is switched over completely to voice yield.

Cambre [2]. This task is chiefly engaged to assist outwardly incapacitated with getting to online entertainment and other web-based administrations, since understanding computerized content is a critical and difficult issue for this client bunch.

Vora, Jash [3]. The framework empowers the client to get breaks given by various applications on a solitary stage. The application will world and give profile the board computerization. This framework performs statically with next to no human mediation.

Deepak Ria [4]. The most natural utilization of telephone is "SIRI" which reasons the end client to illuminate end client versatile with voice and it besides answer the voice charge of the client in this plan open information is gathering thought for creative regulatory creation, basically in the locale old government, biological science and joined endeavor.

Patil Akshay [5]. The proposed framework is planned exclusively by bringing country life into
thought and it will clearly assist with making provincial way of life more agreeable and less tedious. The carried-out abilities will clearly be particularly useful to the client as well as others.

Suresh V Reddy [6]. This paper presents about the plan and execution of Virtual Voice Help and understanding plan content is a critical and difficult issue for this client bunch.

Abhay D [7]. Utilizing the Regular language handling worked in with engineered ability we have finished a shrewd virtual computerized collaborator that can oversee applications, answers to clients' questions, and moreover web look bantering with the human voice.

Gamal Bohouta [8]. This proposition presents the construction of Up-and-coming Age of Virtual Individual Partners that is once more VP As framework intended to speak with a human, with a sound design. This VP As systemhas utilized discourse, designs, video, signals, and different modes for correspondence in both the information and result channel

Dr.Kotrappa Sirbi [9].

PARI has various functionalities of mobile devices like network connection and managing various applications on just the voice commands. Contains key features like Voice Pattern Detection, Keyword Learning, etc. which helpful for end user to use various functionalities and services of the mobile devices

Dhiraj Pratap Singh [10]. In this paper we have examined a Voice Enacted Individual Colleague created utilizing python. This colleague as of now works on the web and performs fundamental undertakings like weather conditions refreshes, stream music, search Wikipedia, open work area applications, and so on.,

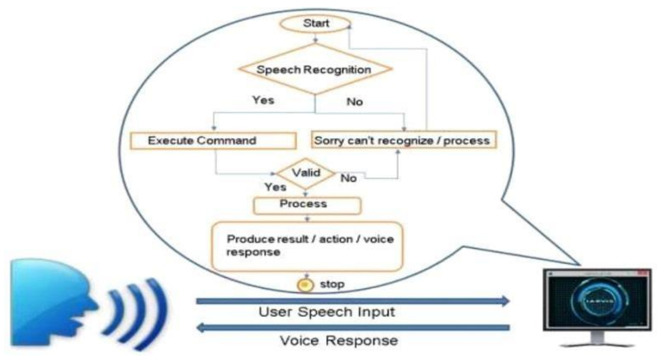
• PROPOSED MODEL

The proposed framework for this model is that every one of the elements of the associate will be performed by utilizing the python modules and the UI is created by utilizing the handyman scoundrel and the face lock which is produced for the security and protection was worked by utilizing man-made brainpower calculation named fisher confronted calculation.

The framework will continue to tune in for orders and the ideal opportunity for listening is variable which can be changed by client necessities. If the framework can't accumulate data from the client input it will continue to ask in the future to rehash till the ideal no. of times

The framework can have both male and female voices as per client necessities. Highlights upheld in the flow rendition incorporate playing music, messages, messages, search on Wikipedia, or opening framework introduced applications, opening anything on the internet browser, and so on. This might be whatever like getting films, opening interior documents, etc. Tests are made through code with the assistance of books and on-line sources, with the plan to track down best outcomes and a more skill of Voice Partner.

Architecture of the proposed model



Prior to beginning the entire high-level undertaking, we have fostered an essential variant of individual right hand which have a few restricted orders.
The above fig shows the essential adaptation engineering. We have done that by utilizing python programming and its modules and a few libraries.

We have involved various modules for various capabilities and activities that are:
PYTTSX3:
To change over text into discourse in python the pyttsx3 module is utilized. This is a disconnected module. The module gives run and stand by usefulness. It is utilized to permit how long the framework will hang tight for one more contribution of client.
DATETIME:
The Date-Time module is imported to help the date and time. Datetime bundle is accustomed to showing Date and Time. This datetime module comes worked in with Python.
WEBBROWSER:
Internet Web browser module is imported to show data from web to clients. To open program and gives input as "Open Google." Then, at that point, input is handled utilizing this module and the Google program is opened. The program which is set in code will open.
WIKIPEDIA:
Wikipedia is a web-based library in python which it workable for the menial helper to deal with the questions on Wikipedia and show it to the clients. This library needs a web association. The quantity of lines that the client needs to come by accordingly can be set physically.
OS MODULE:
Operating system Module gives working framework subordinate functionalities. To perform activities of operating system like information perusing, information composing, or information control ways then these sorts of capabilities are accessible in an operating system module. At the point when these activities raise a mistake like "OSError" in the event of any blunder like invalid names, ways, or contentions which might be wrong or right however not acknowledged by the working framework.
Random:

The irregular module is an inherent module that is utilized to deliver pseudo-arbitrary factors. It could be utilized to do irregular activities, for example, producing an arbitrary number, picking irregular components from a rundown, and rearranging components at arbitrary. For instance, import arbitrary rando seed (2)

SOS module:

This module gives admittance to certain factors utilized or kept up with by the mediator and to capabilities that interface emphatically with the translator.

Pyjokes:

Python upholds production of irregular jokes utilizing one of its libraries. Allow us to investigate it somewhat more, Pyjokes is a python library that is utilized to make one-line jokes for software engineers. Casually, it can likewise be alluded as a great python library which is easy to utilize. This module helps us to extract the jokes from the library and express them to the client.

SMTPLIB module: SMTPLIB is python's standard library which manages messages. The SMTPLIB library sends letters utilizing "SMTP". This is finished utilizing steps that are - initialize, sendmail (), quit.

Speech_recognisation:

The voice module utilized this framework is Google's Programming interface example "import speech_recognition as sr". This module is utilized to perceive the sound waves given by the client as information. This is a free Programming interface this is provided and upheld by Google**.**

Tkinter:

Tkinter module is the standard module of python which is mostly used for graphical User Interfaces.

This is a from TK GUI toolkit.

There are many more frequently used modules for creating the UI's using TK GUI tookit like

- ttk
- tkinter.messagebox
- tkinter.scrolledtext
- tkinter.ttk.Progressbar , etc.,.

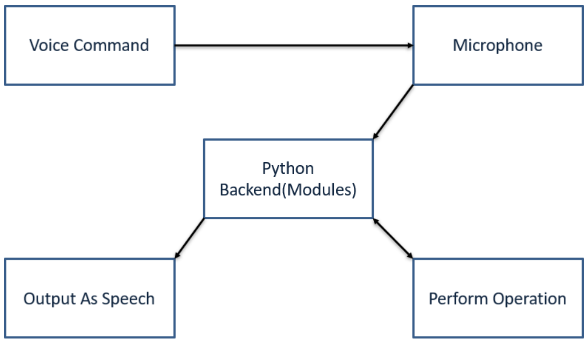
OpenCV:

A computer vision library called OpenCV offers resources for tracking, identifying, and detecting faces. It has already been trained facial recognition and detection models that can be utilised to enable face unlock.

TensorFlow:

Another deep learning framework that may be applied for face recognition is TensorFlow. It features facial recognition models that have already been trained, which may be utilized as a foundation for face unlock.

Block diagram

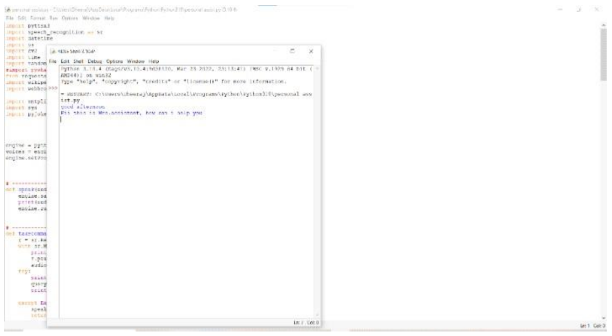


First the client will provide the voice order to the associate utilizing the mouthpiece then the voice will be switched over completely to message utilizing module pyttsx3, then, at that point, the menial helper access the modules as per the voice order given and plays out the activity in the backend and gives the result as a discourse utilizing pyttsx3.

- ## CONCLUSION

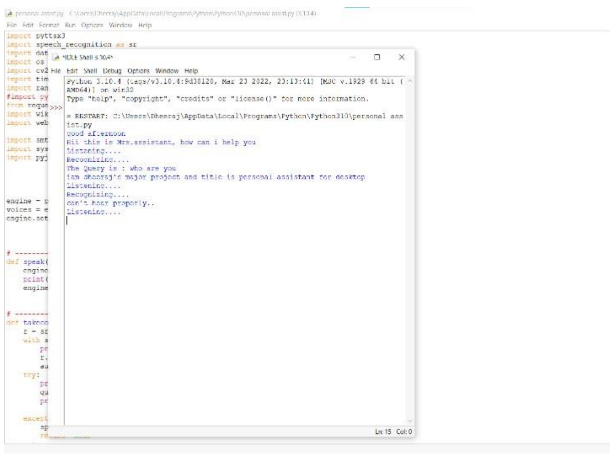
In this "Individual Work area Virtual Voice Asistant utilizing Python" we examined about the plan and execution of Virtual Voice Help. This module utilizes open-source programming py-enchant. The idea of this task makes it adaptable and simple to add extra elements without present day machine functionalities. It is not handing work on human orders anyway moreover it give reactions to the client in view of the inquiry being mentioned or the words spoken via the client comprising of laying out obligations and activities. It is welcoming the way the individual feels more prominent agreeable and to communicate with the voice aide. The utility need to likewise discard any sort of pointless manual artworks expected inside the purchaser presence of acting each test. The total machine works at the verbal enter instead of the resulting one.

- ## SCREENSHOTS



```
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:\Users\[illegible]\AppData\Local\Programs\Python\Python310\personal ass
ist.py
good afternoon
Hi this is Mrs.assistant, how can i help you
```

Fig 5.1

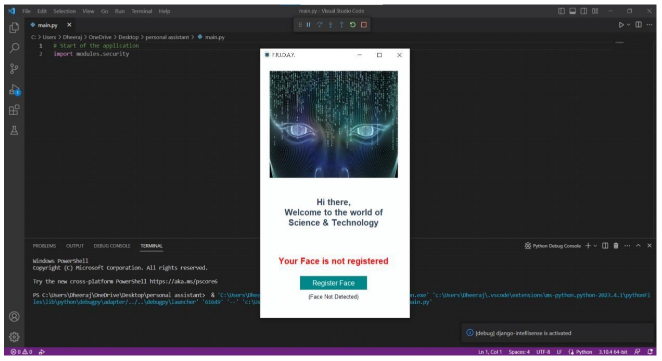


```
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:\Users\Dheeraj\AppData\Local\Programs\Python\Python310\personal ass
ist.py
good afternoon
Hii this is Mrs.assistant, how can i help you
Listening....
Recognizing....
The Query is : who are you
iam dheeraj's major project and title is personal assistant for desktop
Listening....
Recognizing....
can't hear properly..
Listening....
```

Fig 5.2

Fig 5.1 and Fig 5.2 are screenshots of application

Before face recognition



```
# Start of the application
import modules.security
```

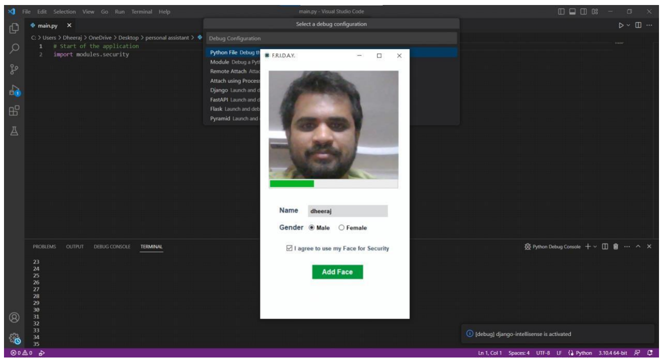
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Dheeraj\OneDrive\Desktop\personal assistant>  & 'C:\Users\Dhee
les\lib\python\debugpy\adapter/../..\debugpy\launcher' '61649' '--' 'c:\Us
```

Hi there,
Welcome to the world of
Science & Technology

Your Face is not registered

(Face Not Detected)

Fig 5.3 Req for face recognition



```
# Start of the application
import modules.security
```

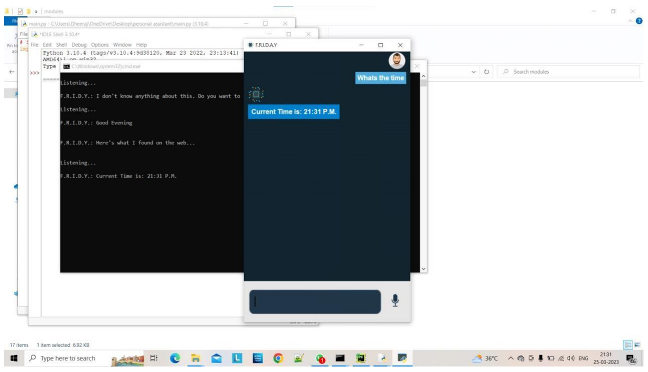
Name: dheeraj

Gender: [✓ Male] Female

☑ I agree to use my Face for Security

F.R.I.D.A.Y.
Select a debug configuration
Debug Configuration
Add Face
[debug] django-intellisense is activated

Fig 5.4 taking user data for recognition



```
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41)
```

```
Listening...

F.R.I.D.Y.: I don't know anything about this. Do you want to

Listening...

F.R.I.D.Y.: Good Evening


F.R.I.D.Y.: Here's what I found on the web...


Listening...

F.R.I.D.Y.: Current Time is: 21:31 P.M.
```

Whats the time

Current Time is: 21:31 P.M.

F.R.I.D.A.Y
main.py - C:\Users\Dheeraj\OneDrive\Desktop\personal assistant\main.py (3.10.4)
IDLE Shell 3.10.4
C:\Windows\system32\cmd.exe
Type here to search

Fig 5.5 chat bot

• Add-on Features

In addition, we've added the user interface to the virtual assistant instead of running the program every required time order to prevent the time consuming and complexity of usage. Even after initializing the user interface, we might face some issues in reaching to the virtual assistant due to the external reasons. For example, if we are using this virtual assistant in a crowdy place and noisy places our virtual assistant might face some issues in recognizing the command which is given by the user, in order to prevent those issues, we installed a chatbot as a new feature by using python modules and tinker cad. By using this we can interact to the virtual assistant easily and efficiently. In keeping the security in mind, we have also installed the face Id lock to the assistant which allows only the device owner to use the assistant.

• References

[1] Jaydeep, Dr, P. A. Shewale, E. Bhushan, A. Fernandes, and R. Khartadkar. "A Voice Based Assistant Using Google Dialog flow and Machine Learning."

International Journal of Scientific Research in Science and Technology 8, no. 3 (2021): 06-17.

[2] Kumar, Lalit. "Desktop Voice Assistant Using Natural Language Processing (NLP)." International Journal for Modern Trends in Science and Technology

(2020): n. pag.

[3] Dr. Kshama Kulhalli, Dr. Kotrappa Sirbi and Mr. Abhijit Patankar, "Personal assistant with Voice Recognition Intelligence", International Journal of Engineering Research and Technology (IJERT), 2017, India

[4] Vora, Jash, Deepak Yadav, Ronak Jain, and Jaya Gupta. "JARVIS: A PC Voice Assistant." (2021).

[5] Pandey, Ankit, Vaibhav Vashist, Prateek Tiwari, Sunil Sikka, and Priyanka Makkar. "Smart Voice Based Virtual Personal Assistants with Artificial

Intelligence." Artificial & Computational Intelligence/Published Online: June (2020).

[6] July 2021| IJIRT | Volume 8 Issue 2 | ISSN: 2349-6002 "Voice Assistant Using Python." Nivedita Singh, Dr. Diwakar Yagyasen, Mr. Surya Vikram Singh,

Gaurav Kumar, Harshit Agrawal

[7] Shende, Deepak, Ria Umahiya, Monika Raghorte, Aishwarya Bhisikar, and Anup Bhange. "AI Based Voice Assistant Using Python." Journal of Emerging

Technologies and Innovative Research 6, no. 2 (2019): 506-509.

[8] Patil, Akshay, Suyash Samant, Mohit Ramtekkar, Shubham Ragaji, and Jayashree Khanapuri. "Intelligent Voice Assistant." In Proceedings of the 3rd

International Conference on Advances in Science & Technology (ICAST). 2020.

[9] Abhay D, Chaitanya K and Rohan K, "Study of Voice Controlled Personal Assistant Device", International Journal of Computer Trends and Technology (IJCTT), December, 2016, India

[10] Cambre, Julia, Alex C. Williams, Afsaneh Razi, Ian Bicking, Abraham Wallin, Janice Tsai, Chinmay Kulkarni, and Jofish Kaye. "Firefox Voice: An Open and

Extensible Voice Assistant Built Upon the Web." In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, pp. 1-18. 2021
[11] Matthew B. Hoy, "Alexa, SIRI, Cortana, and more: An Introduction to Voice Assistants", Medical Reference Services Quarterly Journal,

Jan 12, 2018

[12] Gamal Bohouta, Veton Z Këpuska, "Next - Generation of Virtual Personal Assistants (Microsoft Cortana, Apple Siri, Amazon Alexa, and Google Home)", January-2018, IEEE CCWC 2018, The 8th IEEE Annual Computing and Communication Workshop and Conference, At University of Nevada, Las Vegas, USA.
[13] https://www.voicesummit.ai/blog/how-the-military-is-usingvoice-technology

*[14] Anjali Fapal*1, Trupti Kanade*2, Bharati Janrao*3, Mrunalini Kamble*4, Megha Raule*5 PERSONAL "VIRTUAL ASSISTANT FOR*

WINDOWS USING PYTHON", International Research Journal of Modernization in Engineering Technology and Science

Volume:03/Issue:07/July-2021

*[15] Zeeshan Raza*1, Aftab Amin Sheikh*2, Ankush Shahu*3, Abhijeet Thakur*4." VIRTUAL DESKTOP ASSISTANT USING PYTHON."*

International Research Journal of Modernization in Engineering Technology and Science, Volume:04/Issue:04/April-2022.

[16] Grant Smith, Everything You Need to Ace Computer Science in One Big Fat Notebook, Workman Publishing Co. Inc., March 2020.

[17] By PENG CHENG, Personal Voice Assistant Security and Privacy—A Survey, *Proceedings of the IEEE (Volume: 110, Issue: 4, April 2022).*