ABSTRACT

With the advent of technology, numerous solutions have been developed to aid the visually impaired in using computers effectively. One such innovative solution is the voice-based email system, a desktop application designed for visually impaired individuals to send and read emails seamlessly. This system, built on the Python platform, incorporates 'text to speech' and voice recognition technologies to facilitate email communication. The application enables visually impaired users to interact with emails through auditory feedback, allowing them to perform tasks like composing, sending, and reading emails with ease. By leveraging these technologies, voice-through accessible mail system enhances digital accessibility, ensuring that blind individuals can gain benefit from the internet like everyone else.

Internet accessibility is a crucial aspect of modern life, offering vast amounts of knowledge and information. However, visually impaired people often encounter significant challenges when accessing textual content and online services. Traditional internet websites are usually designed with visual interfaces, making them difficult for blind users to navigate. Recognizing this gap, computer-based accessible systems have emerged to provide alternative means of interaction. Screen readers have been instrumental in enabling blind users to access online content. The Voice based email system builds upon these advancements, offering an efficient and user-friendly solution for email communication.

The Voice based email system architecture is specifically designed to cater to the needs of visually impaired users, ensuring they can access email functionalities easily. By integrating audio feedback and voice commands, the system provides a virtual environment that mimics the capabilities of traditional email clients. Users can dictate emails, listen to incoming messages, and navigate their inbox using voice commands. This techno aspect not only improves accessibility but also empowers visually impaired individuals to manage their digital communication independently. The development of such accessible systems underscores the importance of inclusivity in technology, striving to give equal opportunities and facilities for all users in the digital age.

Keywords: Visually impaired, Voice-based email system, text to speech, voice recognition, internet accessibility, email communication, digital accessibility.

1. INTRODUCTION

1.1 PROJECT DESCRIPTION

The most common email services we use daily are typically not accessible for visually challenged individuals. To make these systems more user-friendly for visually impaired people, various assistive technologies have been developed, including screen readers, automatic speech recognizers, speech- to-text and text-to-speech converters, and Braille keyboards. However, these technologies often fall short of providing the seamless user experience that sighted individuals enjoy. The objective of developing a voice-based email system for the visually impaired is to enable these individuals to access and manage their emails easily and efficiently. The proposed application leverages speech-to- text and text-to-speech technologies, allowing users to control their email accounts solely through voice commands. This system is designed to facilitate the reading, sending, and managing of emails, making email communication more accessible to visually impaired users. By using voice commands, the system can prompt users to perform specific actions, and users can respond verbally, streamlining the email management process which is depicted in fig 1.1.1.

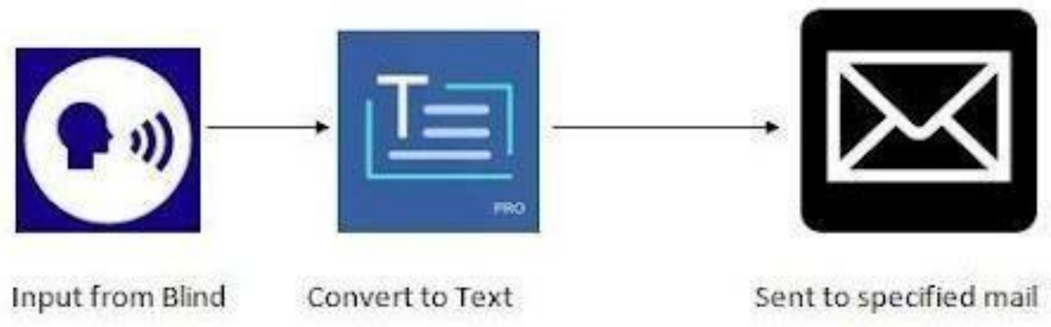


Fig 1.1.1; Working of Voice based Email

This voice-based email system employs the Python to integrate Speech-to-Text (STT) and Text-to-Speech (TTS) technologies effectively. The Speech-to-Text component, also known as Automatic Speech Recognition (ASR), converts spoken language into written text. This feature is crucial for composing emails, as it allows users to dictate their messages instead of typing them. The Text-to-Speech module provides audio feedback, reading out the content of received emails, including the sender's information, the subject line, and the body of the email. This auditory output ensures that visually impaired users can comprehend the email content without needing to see the screen.

Beyond email management, the system aims to assist visually impaired users in navigating and utilizing fundamental applications such as My Computer, Word, and Notepad. By incorporating voice commands, the system allows users to interact with these applications more independently. For instance, users can open and edit documents in Word, create notes in Notepad, and manage files in My Computer, all through voice interactions. This functionality broadens the scope of the system, making it a valuable tool for a wide range of daily tasks.

The implementation of this voice-based email system is expected to significantly enhance the digital accessibility and autonomy of visually impaired individuals. By addressing the limitations of existing assistive technologies, this system strives to provide a more intuitive and user-friendly experience. The integration of STT and TTS technologies ensures that users can perform essential tasks with greater ease and confidence.

In summary, the voice-based email system for the visually impaired represents a significant advancement in assistive technology. By leveraging speech-to-text and text-to-speech converters within the Python packages, the system offers a comprehensive solution for email management and fundamental computer operations. This application not only improves accessibility but also empowers visually impaired individuals to engage with digital communication and computing more effectively. Through continuous development and user feedback, the system aims to evolve and adapt, further enhancing its usability and impact.

2. LITERATURE SURVEY

The development of assistive technology for people with visual impairments has improved considerably, especially in the area of digital communication. The screen reader is among the first and most significant technologies created for this purpose. Screen readers have proven crucial in delivering audio feedback, allowing visually challenged people to navigate text-based content on computers, according to Ahmed and Booth (2014) [1]. However, the requirement to learn intricate keyboard shortcuts and commands has frequently hindered the usability of these technologies, which can reduce accessibility and user happiness.

In their discussion of the advancement of assistive technology for the blind, Shaikh and Patil (2017) in [2] place particular emphasis on the necessity of user-centered design and iterative testing. Their evaluation emphasizes how crucial it is to make these technologies user-friendly and intuitive, which is in line with Story and Maxim's (2016) focus on the integration of text-to-speech (TTS) and speech-to-text (STT) technologies in [3]. For tasks like composing and reading emails, these technologies bridge the gap between spoken language and written text, enabling a more natural contact with digital information.

The incorporation of voice commands into assistive technologies has been a significant step forward. Cooper and Edler (2019) according to [5], highlight how voice commands can dramatically improve the usability of these systems by enabling users to perform tasks with minimal effort. This integration has been facilitated by advances in speech recognition technology, as described by Jiang and Li (2020) in [6], their research points out that modern speech recognition systems are more accurate and reliable, making them well-suited for assistive applications.

Tao and Tan (2017) like in [7], provide a comprehensive review of text-to-speech synthesis from the perspective of assistive technology. They argue that recent improvements in speech synthesis have made it possible to produce more natural and intelligible speech, which is crucial for the robust features of screen readers. Alm and Newell (2018) in [8], support this view, demonstrating that enhanced speech synthesis can significantly improve screen reader usability, leading to a more satisfying user experience.

Voice-based email systems specifically designed for the visually impaired leverage these advancements. Roy and Samanta (2019) as in [9], describe how such systems utilize STT and TTS technologies to enable users to manage their emails through voice commands. This To make email management more accessible and empower visually impaired users by providing greater independence in digital communication, it is crucial to enhance the accuracy of speech recognition systems. According to Liao and He (2020), continuous improvements in these systems are necessary to meet the high standards required for assistive applications. Their research indicates that increased accuracy directly correlates with higher user satisfaction and effectiveness. Technique makes email management more accessible also it empowers visually impaired users by providing them with greater independence in digital communication.

The accuracy of speech recognition systems remains a critical factor in the overall effectiveness of assistive technologies. Liao and He (2020) in [10], evaluate various speech recognition systems and suggest that continuous improvements are necessary to meet the high standards required for assistive applications. Their findings indicate that better accuracy directly correlates with higher user satisfaction and effectiveness.

Designing voice command systems for digital accessibility involves adhering to user-centric design principles, as discussed by Brooks and Peters (2018) in [1]. They emphasize the importance of thorough testing with end-users to ensure that the systems meet the specific needs of visually impaired individuals. This approach ensures that the developed technologies are both functional and user-friendly.

In conclusion, the literature underscores the significance of integrating advanced STT and TTS technologies into assistive applications for visually impaired users. The development of intuitive interfaces that incorporate voice commands has the potential to greatly enhance the usability and accessibility of these technologies. By addressing the challenges identified in earlier research and leveraging recent advancements, the proposed voice-based email system aims to provide a robust solution for email management and basic computer operations, thereby enhancing digital accessibility for visually impaired individuals.

2.1 EXISTING AND PROPOSED SYSTEM

The existing system for voice-based email tailored for blind individuals focuses on facilitating access to email functionalities through voice commands. These systems employ various technologies such as Speech Recognition, Interactive Voice Response (IVR), and mouse click events to enable users to interact with the email client without the need for visual input. The process typically begins with a registration module where users are guided by voice prompts to input their necessary details, thus creating their profile within the system. Following registration, the login module requires users to provide their credentials, which are also managed through voice commands. This design ensures that users who are visually impaired can navigate the system without the necessity of a keyboard, relying instead on auditory feedback and voice interactions. Despite these advancements, the current systems exhibit several significant drawbacks. One of the primary limitations is that these systems generally support only basic email functionalities. Users can send simple text-based emails but are unable to attach files, reply to emails, or save documents. This restriction greatly limits the usability of the email system for more comprehensive and practical communication needs. Moreover, the existing systems lack secure authentication methods. While voice recognition is utilized for user verification, it does not provide a robust security framework, leaving the system vulnerable to potential unauthorized access. Additionally, the absence of audio feedback to read out the contents of the emails further hampers the user experience, as blind individuals heavily rely on auditory information to comprehend the email content.

Furthermore, the reliance on keyboard inputs, even if minimal, poses a challenge for blind users who may find it difficult to recognize and remember keyboard characters. The noisy audio interfaces of screen readers currently in use contribute to the complexity and inconvenience faced by blind users in navigating and operating email systems. These limitations underline the need for a more advanced and user-friendly solution that can comprehensively address the accessibility and usability challenges faced by blind individuals in managing their email communication.

To address the shortcomings of current voice-based email systems for blind users, we propose an advanced and comprehensive solution integrating cutting-edge technologies: The new system incorporates Speech-to-Text (STT) in fig 2., Text-to-Speech (TTS) in fig 2.2, and Interactive Voice Response (IVR) to deliver a seamless and intuitive user experience. The primary goal is to develop an email client that goes beyond basic functionalities, enabling users to perform complex actions such as attaching files, replying to emails, and saving documents. This significantly enhances the system's usability and practicality.

Fig 2.1.1; Speech to Text conversion

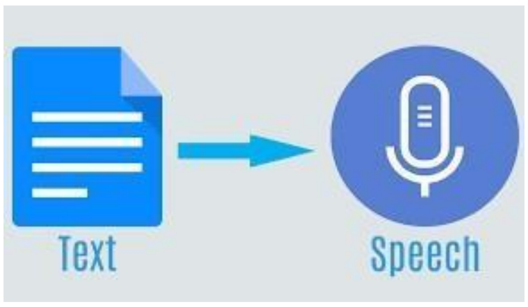


Fig 2.1.2; Text to Speech conversion

A key innovation in this proposed system is the inclusion of a secure face-based authentication mechanism. Utilizing the Haar Cascade classifier (fig 2.3) and Local Binary Patterns Histograms (LBPH) algorithm, this feature ensures a high level of security. By implementing facerecognition for userverification, the system can robustly authenticate users, thereby protecting user data and preventing unauthorized access. This method provides a more secure alternative to voice recognition, addressing a critical security concern in existing systems.

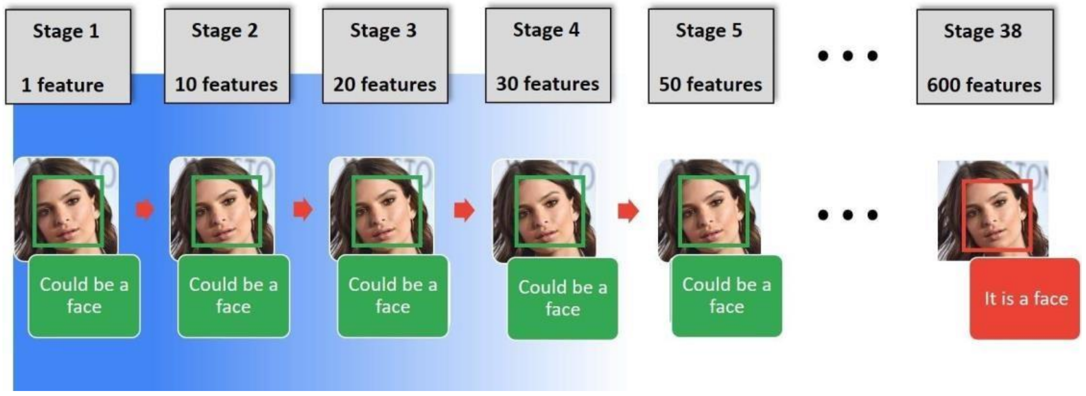


Fig 2.1.3; Working of Haar Cascade Classifier

Improving user interaction through effective auditory feedback is another focus of the proposed system. Advanced TTS technology is employed to read out email contents clearly and accurately, ensuring that blind users can understand their emails without visual input. The use of STT technology allows users to compose and send emails using their voice, completely eliminating the need for keyboard inputs. This approach simplifies the email management process and makes it more accessible to users with varying levels of technological proficiency.

Furthermore, the proposed solution incorporates Multipurpose Internet Mail Extensions (MIME) to support the attachment of files in emails. This feature expands the system's functionality, making it suitable for a broader range of communication needs. By allowing users to send and receive attachments, the system addresses a significant limitation of existing solutions and enhances the overall utility of the email client.

In summary, the proposed voice-based email system for blind individuals aims to provide a comprehensive, secure, and user-friendly solution. By integrating advanced technologies such as face-based authentication, STT, TTS, and MIME, the system seeks to overcome the key limitations of existing systems. This proposed system offers a practical and accessible email management tool that not only meets the basic communication needs of its users but also empowers them with greater control and convenience in managing their email interactions.

2.2 FEASIBILITY STUDY

The feasibility study is a crucial part of the project planning phase. It aims to evaluate the project's potential for success and to assess the practicality of the proposed system. This section covers the technical, operational, and economic feasibility of the voice-based email system for the blind.

2.2.1 Technical Feasibility Study

The technical feasibility study examines whether the technical resources are available and sufficient to support the development and implementation of the voice-based email system for the blind. Key points of consideration include:

- **Technology Availability:** The project leverages existing technologies such as speech recognition, text-to-speech conversion, and email protocols. These technologies are mature and widely supported, ensuring reliability and ease of integration.
- **System Integration:** The integration of various components (voice interface, speech recognition, email server, text-to-speech service, and database) is feasible with current technologies. APIs and SDKs are available for these components, which simplifies the development process.
- **Infrastructure Requirements:** The system requires a stable internet connection for email communication and access to cloud services for speech recognition and text-tospeech conversion. These requirements are common and manageable with current infrastructure capabilities.
- **Scalability and Performance:** The system is designed to handle multiple users and process requests efficiently. Cloud services provide scalability options to accommodate growing user bases and increasing data volumes without significant performance degradation.

2.2.2 Operational Feasibility Study

The operational feasibility study evaluates whether the project can operate effectively within the existing organizational structure and whether it meets the user requirements. Key points of consideration include:

- **User Acceptance:** The target users (visually impaired individuals) will likely find the voice-based email system highly beneficial as it addresses their specific needs. The user-friendly voice interface and audio feedback are designed to be intuitive and easy to use.

- **Training and Support:** Minimal training is required for users to interact with the system. However, providing initial user training sessions and ongoing support will ensure smooth adoption and usage. Help resources such as user manuals and online tutorials can further assist users.

- **Operational Integration:** The system can be integrated into existing workflows with minimal disruption. It complements existing email communication methods and does not require significant changes to current practices.
- **Maintenance and Updates:** Regular maintenance and updates are necessary to ensure the system remains functional and secure. These tasks can be managed with existing IT resources, with additional support from technology providers if needed.

2.2.3 Economic Feasibility Study

The economic feasibility study assesses the financial aspects of the project to determine whether it is economically viable. Key points of consideration include:

- **Initial Investment:** The initial costs include software development, infrastructure setup, and user training. These costs are offset by the availability of open-source technologies and cloud-based services, which reduce the overall expenditure.
- **Operational Costs:** Ongoing operational costs include server maintenance, cloud service subscriptions, and support services. These costs are predictable and manageable within the organization's budget.
- **Cost-Benefit Analysis:** The benefits of the system, such as improved accessibility for visually impaired individuals, increased productivity, and enhanced user satisfaction, outweigh the costs. The system can also open new opportunities for collaboration and communication, providing additional value.
- **Funding and ROI:** Potential funding sources include government grants, non-profit organizations, and corporate sponsorships. The return on investment (ROI) is realized through improved user engagement, positive social impact, and potential cost savings from increased efficiency.

Overall, the feasibility study demonstrates that the voice-based email system for the blind is technically viable, operationally effective, and economically beneficial. The project has a high likelihood of success and provides significant value to its users.

2.3 TOOLS AND TECHNOLOGIES USED

2.3.1 Python

Python is a high-level, interpreted programming language known for its simplicity and readability. Developed by Guido van Rossum in the late 1980s, Python has gained immense popularity due to its versatility, ease of learning, and strong community support. It has become one of the preferred languages for a wide range of applications, including web development, scientific computing, artificial intelligence, and automation.

Advantages of Python

- **Simplicity and Readability:** Python emphasizes readability with its clean and straightforward syntax. Its code readability makes it easier for developers to write and maintain code, reducing the time and effort required for development and debugging.
- **Versatility:** Python supports multiple programming paradigms, including procedural, object- oriented, and functional programming styles. It offers a flexible environment suitable for diverse application domains, from web development to data science and automation.
- **Large Standard Library:** Python comes with a comprehensive standard library that provides modules and functions for performing various tasks without needing third-party libraries. This standard library includes modules for file I/O, networking, GUI development, data manipulation, and more.
- **Community and Support:** Python has a vibrant and active community of developers and enthusiasts. The community contributes to the language's growth by developing libraries, frameworks, and tools. It also provides extensive documentation and support through forums, tutorials, and online resources.
- **Cross-platform Compatibility:** Python runs on various platforms, including Windows, macOS, and Linux. This cross-platform compatibility allows developers to write code once and run it on different operating systems without modifications.
- **Integration Capabilities:** Python supports seamless integration with other languages and tools, making it ideal for building complex systems and integrating with existing infrastructure. It can interface with C/C++, Java, and .NET components, facilitating interoperability.

2.3.2 Python Libraries

• **Tkinter**: Tkinter is the standard GUI package for creating graphical desktop applications in Python. This project utilizes Tkinter to design and implement the main user interface (UI), incorporating windows, buttons, labels, and input fields needed for user registration and email system interaction.

• **OpenCV (Open-source Computer Vision Library)**: OpenCV is a robust computer vision library used for object detection and real-time image processing. In this project, OpenCV handles facial detection and recognition tasks. During user registration, the system captures images using a webcam, processes them for facial authentication, and utilizes Haar cascades for face identification.

• **Pandas**: Pandas is a data manipulation library that provides data structures and tools for working with structured data. This project employs Pandas to read and write user registration details to CSV files. It manages tabular data containing user information, facilitating data handling and storage operations.

• **MySQLdb**: MySQLdb is a MySQL database interface for Python, enabling interaction with MySQL databases from Python applications. This project integrates MySQLdb to store and manage user registration data in a MySQL database. It executes SQL queries, inserts new records, and manages database connections seamlessly.

• **gTTS (Google Text-to-Speech)**: gTTS is a library for converting text into speech using Google's Text-to-Speech API. In this project, gTTS generates audio feedback and prompts for users during various stages of the application, enhancing user interaction with spoken notifications and instructions.

• **SpeechRecognition**: SpeechRecognition is a library for performing speech recognition in Python, supporting multiple speech recognition engines. This project uses SpeechRecognition to capture spoken commands from the user, enabling voice-based interaction with the email system. Users can compose emails and perform other operations using voice commands.

• **NumPy**: NumPy is a fundamental library for numerical computing in Python, providing support for large arrays and matrices. This project uses NumPy for numerical operations and image processing tasks. It converts images into NumPy arrays, manipulates image data, and performs calculations necessary for image-based operations.

• **Pillow (PIL)**: Pillow is a fork of the Python Imaging Library (PIL) that supports various image processing tasks. In this project, Pillow handles image files by opening, manipulating, and saving them in different formats. It also converts images to grayscale for facial detection and recognition purposes.

• **OS Module**: The OS module in Python provides functionalities for interacting with the operating system, such as managing files and directories. In this project, the OS module is used to manage file paths, create directories, and delete temporary files generated during image and audio processing, ensuring efficient file management.

• **CSV Module**: The CSV module in Python offers classes for reading and writing data in CSV format. This module is used in the project to manage user registration data. It reads user details from CSV files, writes new records, and handles tabular data operations, ensuring accurate storage and retrieval of user information.

Python's simplicity, versatility, extensive standard library, and robust community support make it a preferred choice for developing a wide range of applications, including the Voice-Based Email System demonstrated in this project. By leveraging Python and its relevant libraries, developers can create efficient, scalable, and user-friendly applications that integrate advanced functionalities such as GUIs, computer vision, database interactions, and speech recognition seamlessly.

2.3.3 MySQLDatabase

MySQL is a widely-used open-source relational database management system (RDBMS) renowned for its reliability, scalability, and ease of use. It is commonly employed in web applications and various software for storing and managing structured data.

- Purpose: MySQL serves as the primary database management system for storing user registration details and managing authentication data in the Voice-Based Email System. It offers a structured and efficient method for securely storing user information.
- Integration: The project integrates MySQLdb, a Python interface for MySQL, to facilitate connections between the Python application and the MySQL database. This integration allows the application to execute SQL queries, insert new records, and retrieve data from the MySQL database seamlessly.
- Functionality: MySQL enables the creation of tables to store user credentials, such as usernames, hashed passwords, names, emails, encrypted passwords, mobile numbers, and

unique identifiers (UIDs). It ensures data integrity and supports efficient retrieval of user information during login and registration processes.

2.3.4 SQL (Structured Query Language)

SQL is a standardized programming language used for managing relational databases. It provides a set of commands (queries) to perform operations like data insertion, retrieval, updating, and deletion in relational database systems.

Usage: In the project, SQL queries are extensively used to interact with the MySQL database. These queries are executed through Python using MySQLdb to perform various tasks such as:

- Inserting new user records into the database upon successful registration.
- Verifying the existence of usernames to prevent duplicates during registration.
- Retrieving user information based on credentials for authentication purposes.
- Managing and updating user records as needed by the application's functionalities.

Benefits: SQL ensures the reliability and consistency of data operations within the database. It enables efficient data manipulation and retrieval, supports transactions to maintain data integrity, and allows developers to implement complex database interactions required by the Voice-Based Email System.

2.3.5 MySQL Workbench

MySQL Workbench is a graphical tool used to design, develop, and administer MySQL databases. Itprovides a visual interface for database administrators and developers to manage database schemas, execute SQL queries, and perform database maintenance tasks.

Role: MySQL Workbench is used for designing and visualizing the database schema for the Voice- Based Email System. It allows developers to create tables, define relationships between entities, andmanage database configurations efficiently.

Features: With MySQL Workbench, programmers can exhibit various tasks such as:

- Creating and modifying database schemas using a visual data modelling tool.
- Writing and executing SQL queries directly within the interface for testing and debugging.
- Managing database connections and server configurations to ensure optimal performance and security.

Advantages: MySQL Workbench enhances productivity by providing a user-friendly environment for database design and administration. It facilitates team work and unity among team collegues by enabling them to visualize and modify the database structure seamlessly.

The integration of MySQL and SQL in the Voice-Based Email System project demonstrates the effective use of database tools and technologies to manage user registration data securely and efficiently. By leveraging MySQL and SQL capabilities through Python's MySQLdb interface, the application ensures reliable data storage, retrieval, and manipulation, supporting seamless user interaction and authentication processes. This approach underscores the importance of robust database management tools in developing scalable and reliable software applications.

2.3.6 Machine Learning Algorithms

Haar Cascade for Face Detection

Haar Cascade is an object detection algorithm used to identify objects in images or videos. Developed by Paul Viola and Michael Jones, it was introduced in their 2001 paper "Rapid Object Detection using a Boosted Cascade of Simple Features."

Algorithm Overview: Haar Cascade operates through several stages, each comprising multiple weak classifiers known as decision stumps. These classifiers are trained with a machine learning method called AdaBoost, which selects the most significant features (Haar-like features) from a vast pool of candidates.

Steps Involved

1. **Haar Feature Selection:** Identifying key features in images, such as edges, lines, and contrasts as shown in fig 2.3.1.

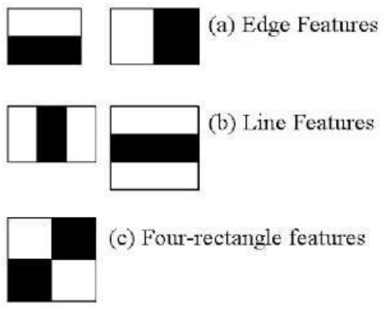


Fig 2.3.1; Selection of Haar feature

2. **Integral Images:** Calculating integral images to expedite the feature evaluation process as shown in fig 2.3.2.

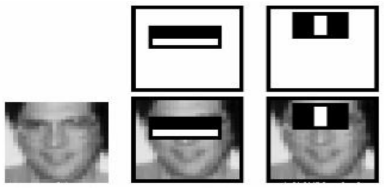


Fig 2.3.2; Calculation of Integral images

3. Adaboost Training: Choosing the optimal features and training classifiers based on these selected features. The training process is illustrated in Figure 2.3.3.

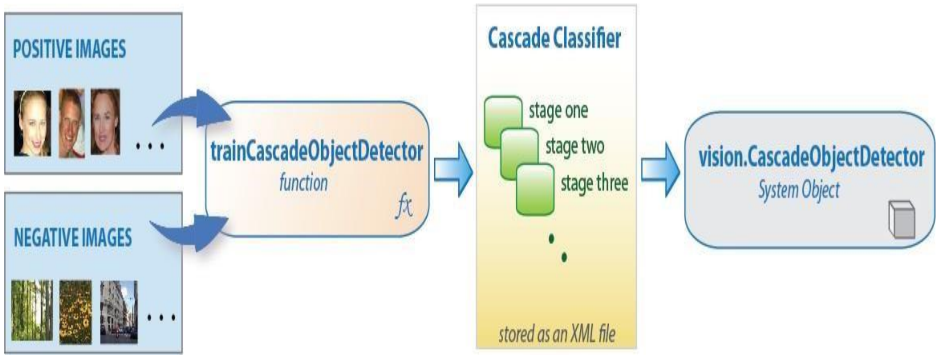
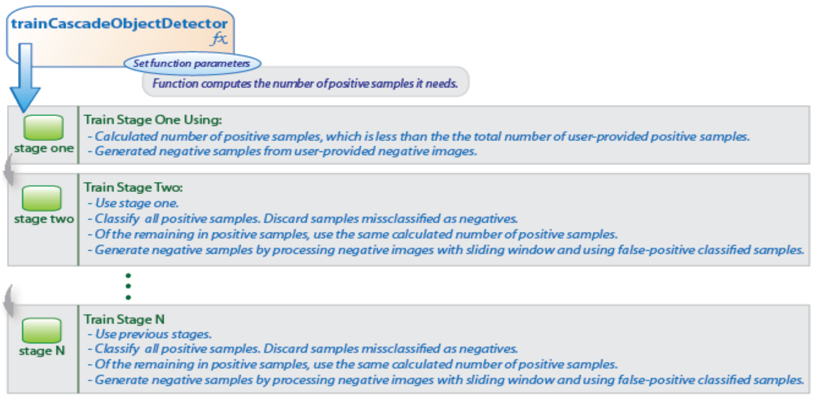


Fig 2.3.3; Adaboost training mechanism

4. Cascade Classifiers: Organizing classifiers into stages to create a robust classifier cascade that effectively identifies objects while reducing false positives. The classifier's structure is detailed in Figure 2.3.4.



Function computes the number of positive samples it needs.

Train Stage One Using:
- Calculated number of positive samples, which is less than the the total number of user-provided positive samples.
- Generated negative samples from user-provided negative images.

Train Stage Two:
- Use stage one.
- Classify all positive samples. Discard samples missclassified as negatives.
- Of the remaining in positive samples, use the same calculated number of positive samples.
- Generate negative samples by processing negative images with sliding window and using false-positive classified samples.

Train Stage N
- Use previous stages.
- Classify all positive samples. Discard samples missclassified as negatives.
- Of the remaining in positive samples, use the same calculated number of positive samples.
- Generate negative samples by processing negative images with sliding window and using false-positive classified samples.

Fig 2.3.4; working of cascade classifiers

Application

- **Data Collection:** Collecting a dataset that includes positive images (faces) and negative images (non-faces).
- **Face Detection:** Identifying faces in real-time camera feeds for authentication during login.

LBPH Algorithm for Face Recognition

Local Binary Patterns Histogram (LBPH) is a texture descriptor used for facial recognition tasks, known for its simplicity and effectiveness.

Algorithm Description

- **Local Binary Patterns:** Computes a binary pattern by comparing each pixel with its neighboring pixels within a specified radius.
- **Histogram Calculation:** Constructs histograms of the local binary patterns extracted from different regions of the face.
- **Feature Vector:** Represents each face with a feature vector created by concatenating the histograms of all regions.
- **Classification:** Determines identity by comparing the feature vectors of test images with those stored in the database using distance metrics like Euclidean distance or Chi-square.

Steps involved

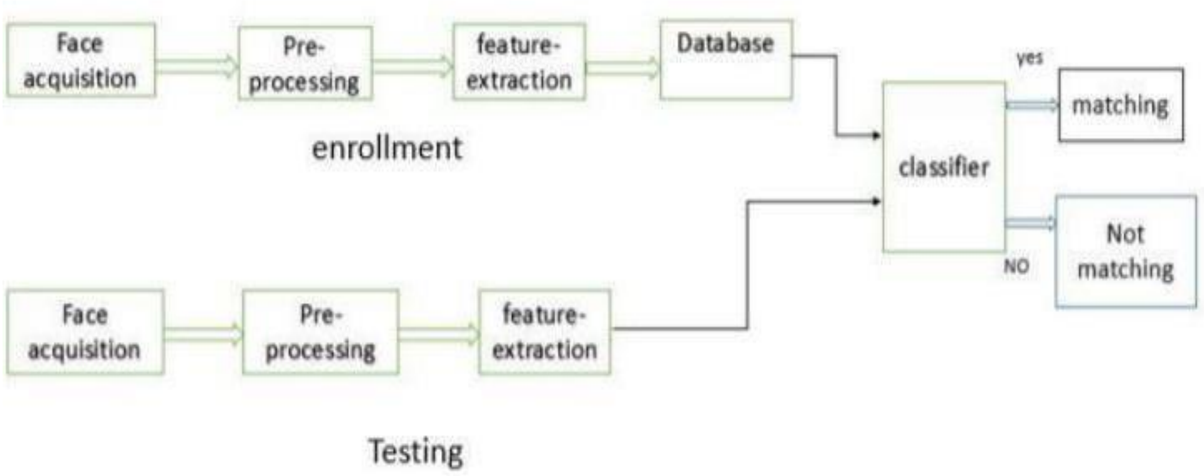


Fig 2.3.5; Steps involved in Algorithm

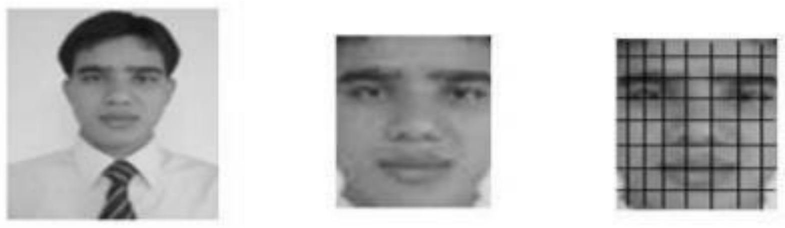


Fig 2.3.6; Dividing dimensions

The LBP operator is applied to every region. As shown in Figure 2.3.7, the LBP operator is defined within a 3x3 window.

$$\mathrm{LBP}(x_c, y_c) = \sum_{p=0}^{P-1} 2^p s(i_p - i_c)$$

Fig 2.3.7; LBP Operator

In this context, '(Xc, Yc)' represents the central pixel with intensity 'Ic'. 'In' denotes the intensity of a neighboring pixel. The algorithm uses the median pixel value as a threshold, comparing each pixel to its eight closest neighbors using the function illustrated in Figure 2.3.8.

$$s(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

Fig 2.3.8; Median pixel function

- f a neighboring value is greater than or equal to the central value, it is assigned a value of 1; otherwise, it is assigned a value of 0.
- This process yields 8 binary values from the 8 surrounding neighbors.
- Combining these binary values results in an 8-bit binary number, which can be converted to a decimal number for easier interpretation.
- This decimal number, as shown in Figure 2.3.9, represents the pixel's Local Binary Pattern (LBP) value, which ranges from 0 to 255.

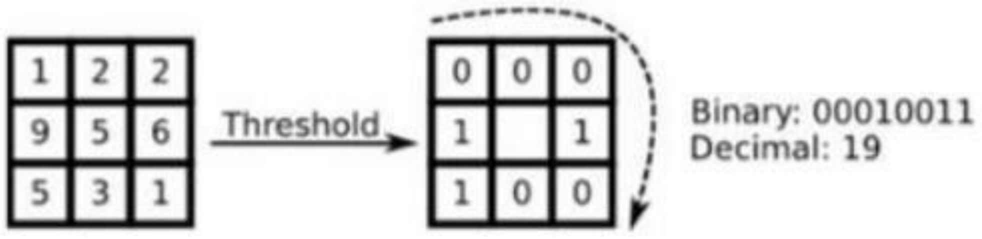


Fig 2.3.9; Pixel LBP value

It was later observed that a fixed neighborhood did not capture details varying in scale effectively. The algorithm was enhanced to incorporate varying radii and numbers of neighbors, leading to the development of Circular Local Binary Patterns (LBP), as illustrated in Figure 2.3.10.

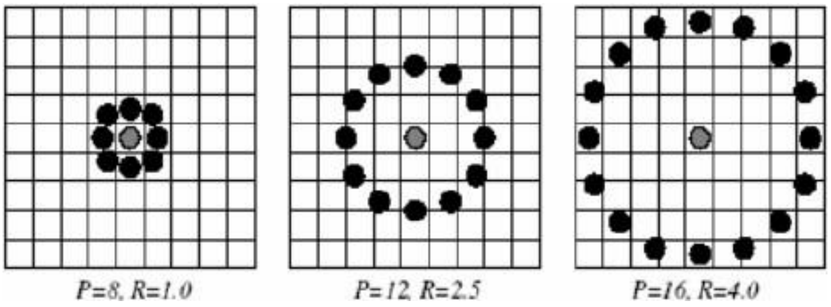


Fig 2.3.10; Circular LBP

- The concept involves arranging a variable number of neighbors on a circle with an adjustable radius. This approach captures the following neighborhoods:

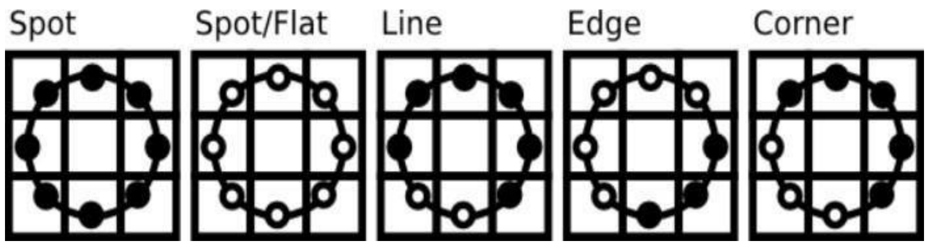


Fig 2.3.11; Capturing of Neighbourhood

- For a given point (Xc, Yc), the position of a neighbor (Xp, Yp), where p belongs to set P, can be determined using the equations illustrated in Figure 2.3.12.

$$x_p = x_c + R\cos\left(\frac{2\pi p}{P}\right)$$
$$y_p = y_c - R\sin\left(\frac{2\pi p}{P}\right)$$

Fig 2.3.12; Calculation of p

- Here, R represents the radius of the circle, and P denotes the number of sample points.
- If the coordinates of points on the circle do not align with the image coordinates, they are typically interpolated using bilinear interpolation.

$$f(x,y) \approx \begin{bmatrix} 1-x & x \end{bmatrix} \begin{bmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{bmatrix} \begin{bmatrix} 1-y \\ y \end{bmatrix}$$

Fig 2.3.13; Bilinear interpolation

.

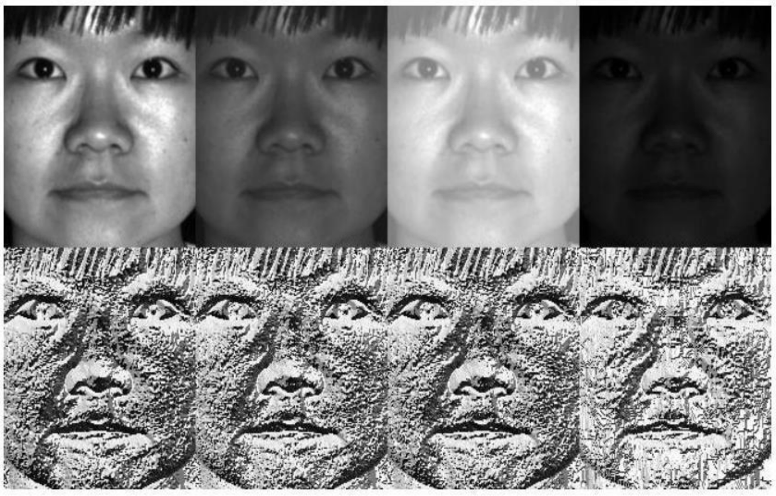


Fig 2.3.14; Monotonic gray scale transformations

- Once the LBP values are generated, a histogram of the region is created by counting the occurrences of each LBP value, as shown in Figure 2.3.15.
- After constructing histograms for each region, they are combined into a single histogram, which is referred to as the feature vector of the image.

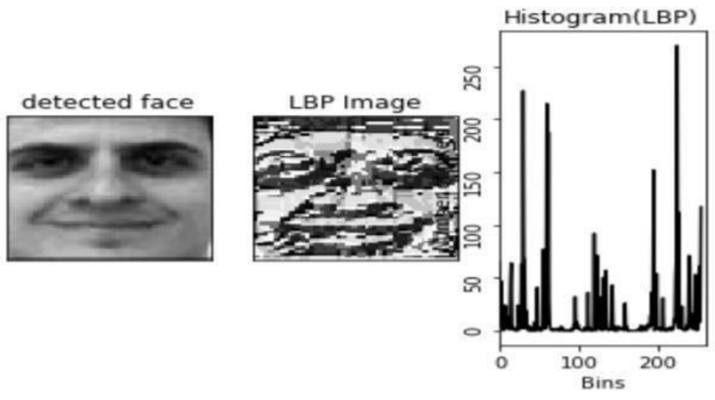


Fig 2.3.15; Comparison of histograms of test image and database image

- Next, we compare the histogram of the test image with those in the database and return the image with the most similar histogram. This comparison can be done using various techniques such as Euclidean distance, chi-square, or absolute value.
- The Euclidean distance (illustrated in Figure 2.3.16) is calculated by comparing the features of the test image with those stored in the dataset. The smallest distance between the test image and an original image indicates the best match.

$$d(a,b) = \sqrt{\sum_{i=1}^{n} \left|a_i - b_i\right|^2}$$

Fig 2.3.16; Euclidian distance

As an output we get an ID of the image from the database if the test image is recognized.

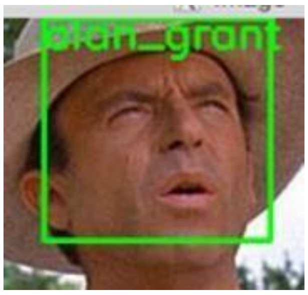


Fig 2.3.17; Image is recognized

LBPH can recognize both side and front faces and it is not affected by-illumination variations which means that it is more-flexible as shown in fig 2.3.17.

Advantages

- **Robustness:** LBPH is robust against illumination changes and facial expressions, making it suitable for varying environmental conditions.
- **Flexibility:** It can recognize faces in different poses and orientations, enhancing its practical usability. **Applications**
- **Building and Training:** Used to train a face recognition model on a dataset collected via web camera.
- **Face Recognition:** Implemented during user login to authenticate users based on facial features captured via the camera.

The integration of Haar Cascade for face detection and LBPH algorithm for face recognition in the project underscores the utilization of robust machine learning techniques for enhancing the security and usability of the Voice-Based Email System. These technologies enable accurate detection and recognition of faces, facilitating secure user authentication and interaction with the email application.

2.4 HARDWARE AND SOFTWARE REQUIREMENTS

2.4.1 Hardware Requirements

- Computer System: Intel Core i5 or equivalent (8th generation or higher recommended)
- Memory: 8 GB RAM (16 GB recommended)
- Storage: Minimum 256 GB SSD
- Web Camera: HD webcam
- Microphone: Integrated or external microphone
- Speakers or Headphones: Required for audio feedback

2.4.2 Software Requirements

- Operating System: Windows 10 or above(64-bit)
- Python Version: Python 3.9
- API: Google's Gmail API
- Speech Recognition Libraries: Python Speech Recognition library
- Text-to-Speech Libraries: Google Text-to-Speech API, pyttsx3
- IDE: Integrated Development Environment (e.g., PyCharm, Visual Studio Code)
- Virtual Environment: Virtualenv or Anaconda for managing dependencies • Internet: Stable Internet Connection

3. SOFTWARE REQUIREMENT SPECIFICATION

3.1 USERS

- The Voice-Based Email System is designed to cater to a diverse range of users, particularly focusing on those with visual impairments or those who prefer hands-free interaction with digital devices. This system aims to be inclusive and accessible, ensuring that it can be used by individuals of varying technical proficiency levels.
- **Primary Users:**

  • **Visually Impaired Individuals:** Users who have partial or complete vision loss and rely on auditory feedback for digital interactions.

  • **Elderly Users:** Older adults who may find traditional interfaces challenging to use and prefer voice-based commands for ease of use.

  • **Busy Professionals:** Individuals who need to manage emails efficiently while multitasking, allowing them to perform email operations hands-free.

  • **Technologically Novice Users:** Individuals who are not well-versed with complex digital interfaces but can interact using simple voice commands.

3.1.1 Scope and Objective

- The Voice-Based Email System aims to provide a user-friendly interface for accessing and managing email through voice commands. The primary objectives include:
- Voice Interaction: Allow users to interact with the system using natural language commands converted from speech to text.
- Email Management: Enable users to perform basic email operations such as reading inbox messages, composing new emails, and managing drafts.
- User Authentication: Implement face recognition for secure user login using Haar Cascade for face detection and LBPH algorithm for face recognition.
- Accessibility: Ensure users with visual impairments or those who prefer hands-free interaction.

3.1.2 Assumptions and Dependencies

Assumptions

- Users have access to a compatible web camera for face authentication.

- The system assumes reliable internet connectivity for interacting with email services via APIs.
- Users are comfortable with basic voice commands and can articulate commands clearly for effective system interaction.

Dependencies:

- **Hardware Dependencies:** Requires a computer system with a web camera, microphone, and internet connectivity.
- **Software Dependencies:** Relies on Python programming language, OpenCV for image processing, Gmail API for email interactions, and speech-to-text and text-to-speech APIs for voice command processing.

3.2 FUNCTIONAL REQUIREMENTS

This specify the system's operations and capabilities:

Data Collection:

- Collect at least 200 face images per user for training the face recognition model.
- Store collected images securely in a designated folder.

Building and Training of Face Recognition Model:

- Utilize Haar Cascade algorithm for initial face detection from collected images.
- Implement LBPH algorithm to train the face recognition model based on detected faces.
- Generate a trained model file (.yml) for user authentication.

Face Recognition:

- Authenticate users during login by capturing and processing real-time facial data.
- Compare captured face features with the trained model to authorize the email system.
- Grant the email interface upon successful authentication; otherwise, indicate invalid user credentials.

Mail Interaction:

- Registration: Allow users to register by providing necessary details through voice commands.
- Login: Enable users to log in using their username and password via speech-to-text conversion.
- Email Operations: Provide functionality to read inbox messages, compose new emails, and logout using voice-guided interactions.

3.3 NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements outline the system's quality attributes:

Performance:

- **Response Time:** Ensure rapid response times for face detection and recognition, typically within a few seconds.
- **Accuracy:** Strive for high accuracy in face recognition to minimize false positives and negatives.

Security:

- **Data Security:** Encrypt stored face images and user credentials to prevent unauthorized access.
- **Authentication:** Utilize secure authentication mechanisms based on face recognition to enhance system security.

Usability:

- **Accessibility:** Make the system accessible for visually impaired users through voice-based interactions.
- **User Interface:** Design an intuitive interface with voice prompts to guide users for seamless navigation.

Reliability:

- **System Availability:** Ensure high availability of the email system, minimizing downtime to allow consistent user access.

- **Error Handling:** Implement robust error handling to manage exceptions and maintain system stability.

Compatibility:

- **Platform Compatibility:** Ensure the system is compatible with major operating systems (Windows, Linux) and web browsers for consistent performance.

The Software Requirements Specification (SRS) for the Voice-Based Email System details the user expectations, functional capabilities, and quality attributes essential for its development and deployment. By addressing these requirements comprehensively, the project aims to deliver a secure, accessible, and user-friendly email management system optimized for voice-based interactions.

4. SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

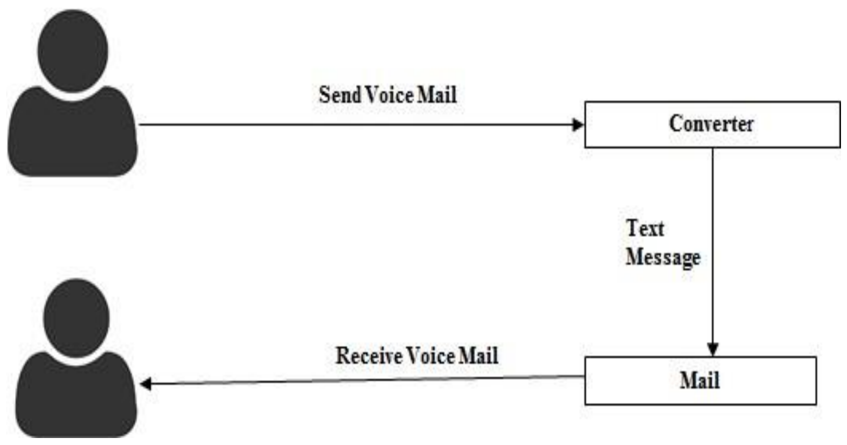


Fig 4.1; System Architecture

The above fig 4.1 illustrates a voice-based email System Architecture designed for visually impaired users. The system aims to facilitate email communication through voice commands, making it accessible for those who are blind or have severe vision impairments.

The process begins with the sender recording a voice mail. This voice mail is then sent to the "Converter" component, which transforms the voice mail into a text message using speech-to-text technology. The Converter recognizes and transcribes the spoken words into written text format. After conversion, the text message is sent to the "Mail" component. This component functions like a traditional email system, processing, storing, and preparing the text message for delivery to the recipient. The recipient, also visually impaired, receives the email as a voice mail. This indicates that the system can convert text messages back into voice messages for the recipient to listen to speaker.

Overall, this architecture integrates speech recognition and speech synthesis technologies to create an inclusive email communication system, allowing visually impaired users to independently send and receive emails, thus enhancing their accessibility to digital communication tools.

4.2 SYSTEM PERSPECTIVE

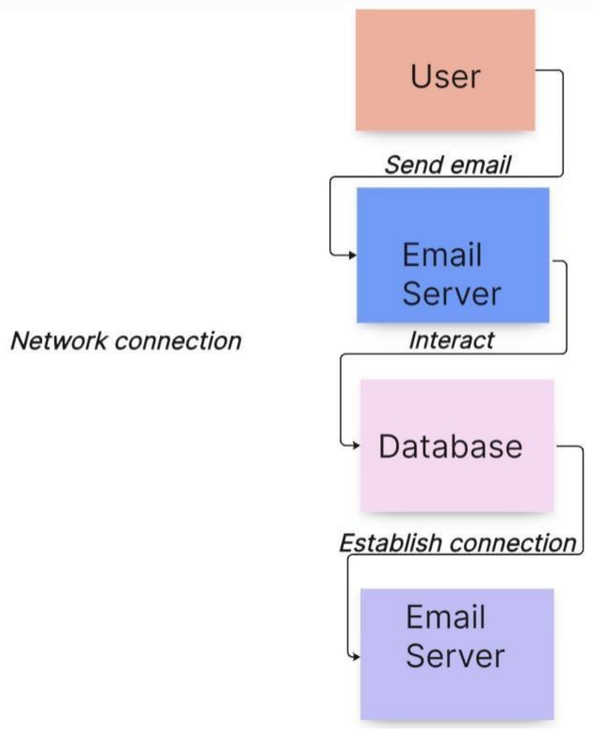


Fig 4.2; System perspective

This simplified system perspective diagram the fig 4.2 for a voice-based email system illustrates the key components and their interactions. Here's an explanation of each part and the flow of actions:

- **User**: Sends an email and the user interacts with the system through a voice interface. For simplicity, this interaction is represented as the user sending an email.
- **Email Server** (First Instance): Receives the email from the user and the email server interacts with the database to store the email and other necessary information.
- **Database**: Stores the email and other related data and database communicates with the email server to save the email information.
- **Email Server** (Second Instance): Establishes a connection for network communication and the email server connects to an external email server or network to send the email to the recipient.

4.2 CONTEXT DIAGRAM

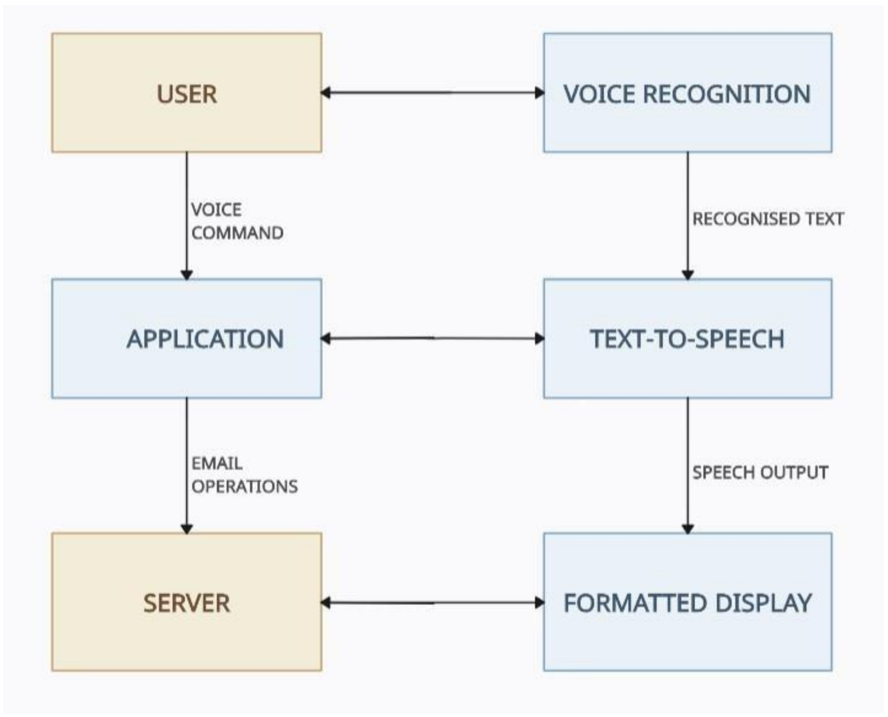


Fig 4.3; Context diagram

The context diagram fig 4.3 represents a voice-based email system tailored for blind users, detailing the essential components and their interactions.

- **User:** The user interacts with the system using voice commands, which is essential for blind users relying on auditory input.
- **Voice Recognition:** This component converts the user's voice commands into text, allowing the system to interpret and process these commands.
- **Application:** The application serves as the system's core, handling email-related operations based on the text commands received.
- **Text-to-Speech:** Converts text responses from the application into spoken words, providing auditory feedback to the user.
- **Server:** Manages email data, handling requests from the application for retrieving, sending, and storing emails.
- **Formatted Display:** Provides a visual representation of email operations for sighted users or administrative purposes.

5. DETAILED DESIGN

5.1 DATAFLOW DIAGRAM

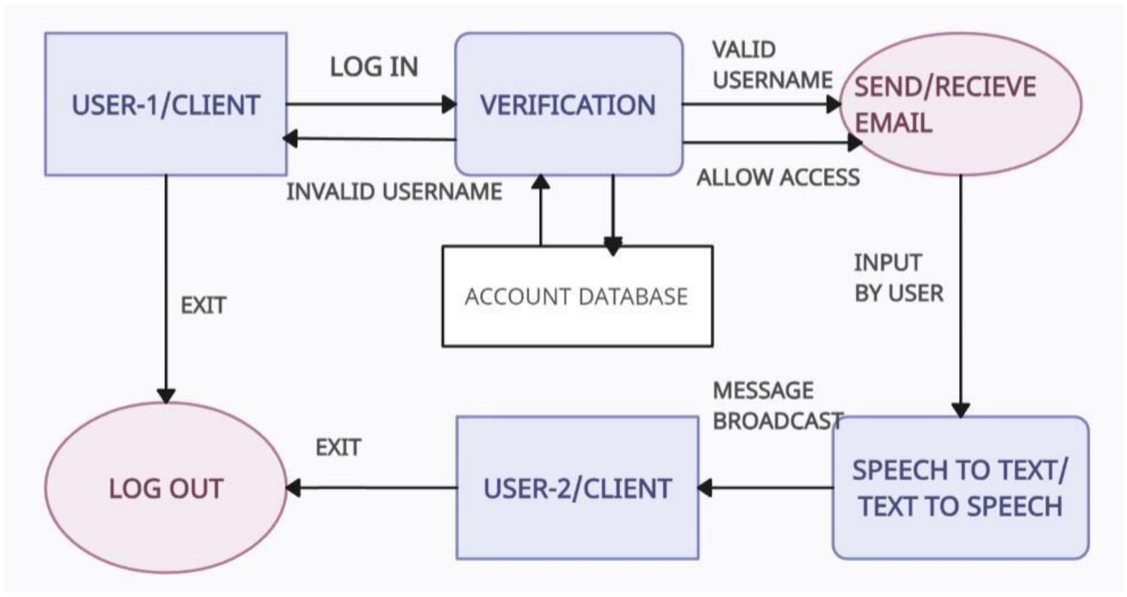


Fig 5.1; Data flow diagram

This Data Flow Diagram (DFD) fig 5.1 is Level 1 which illustrates the detailed processes and data interactions in a voice-based email system designed for blind users.

- **User-1/Client:** The user initiates the process by attempting to log in. This input is directed to the verification process.
- **Verification:** This component checks the credentials against the Account Database.
- **Send/Receive Email:** Upon successful verification, the user gains access to email functionalities, including sending and receiving emails. The user inputs commands to manage emails.
- **Speech to Text/Text to Speech:** This component converts the user's spoken commands into text for processing and converts text responses into speech for the user. It facilitates seamless interaction for blind users.
- **User-2/Client:** Another user who receives the broadcasted messages, facilitating communication between users.
- **Log Out:** Both users can exit the system using the logout process, ensuring the session is terminated securely.

5.2 ACTIVITY DIAGRAM

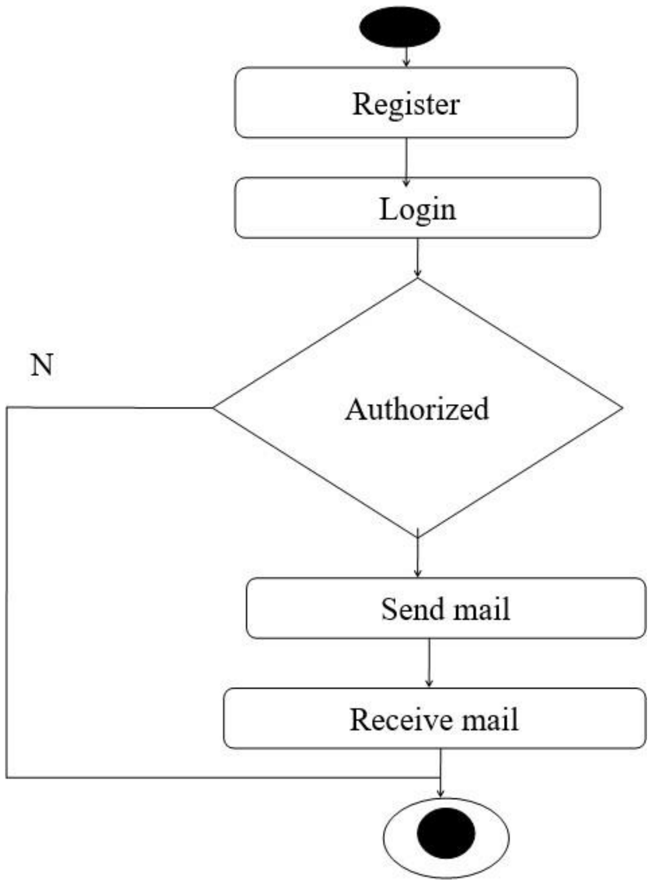


Fig 5.2; Activity diagram

The activity diagram in Figure 5.2 demonstrates the workflow of a voice-based email system designed for visually impaired users. The process starts with user registration, where an account is created. After registering, the user logs into the system.

Upon logging in, the system verifies the user's authorization. If the user is not authorized (indicated by "N"), they are redirected to the login step. If authorized, the user can proceed to send emails. Following this, the user can receive emails. This sequence ensures that users can only access the email functionalities (sending and receiving) after completing the registration, login, and authorization verification steps. The diagram highlights the iterative nature of authorization checks, thereby maintaining a secure communication process for visually impaired users.

5.3 SEQUENCE DIAGRAM

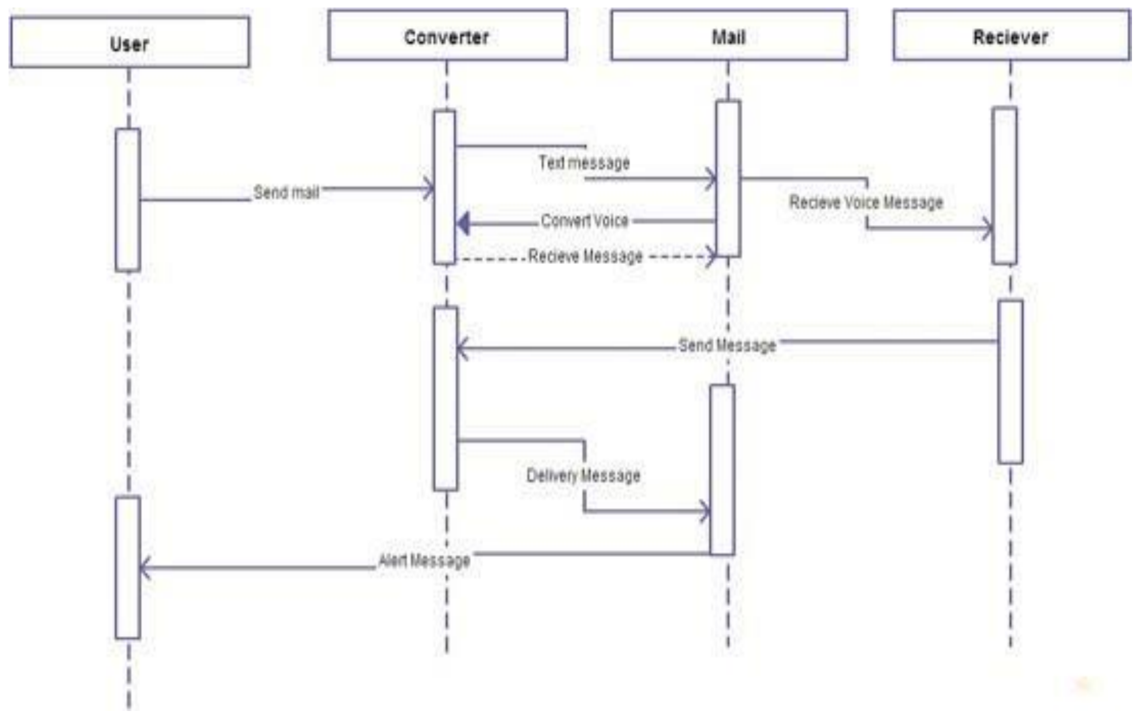


Fig 5.3; Sequence diagram

The above diagram fig 5.3 illustrates the objects interaction with one another in a particular sequence of time. Components are:

- **User**: The sender of the email.
- **Converter**: A system component that converts voice messages to text.
- **Mail**: The email system that sends and receives messages.
- **Receiver**: The recipient of the email.

A user sends a mail, which is first received by the Converter. The Converter converts the voice message to text and forwards it to the Mail system. The Mail system sends the text message to the Receiver. The Receiver gets the voice message, and the Mail system sends a delivery confirmation back to the Converter. Finally, the Converter alerts the user their email has been delivered. This sequence illustrates the interaction and conversion process in a voice-based email system, ensuring successful communication and confirmation at each step.

5.4 USE CASE DIAGRAM

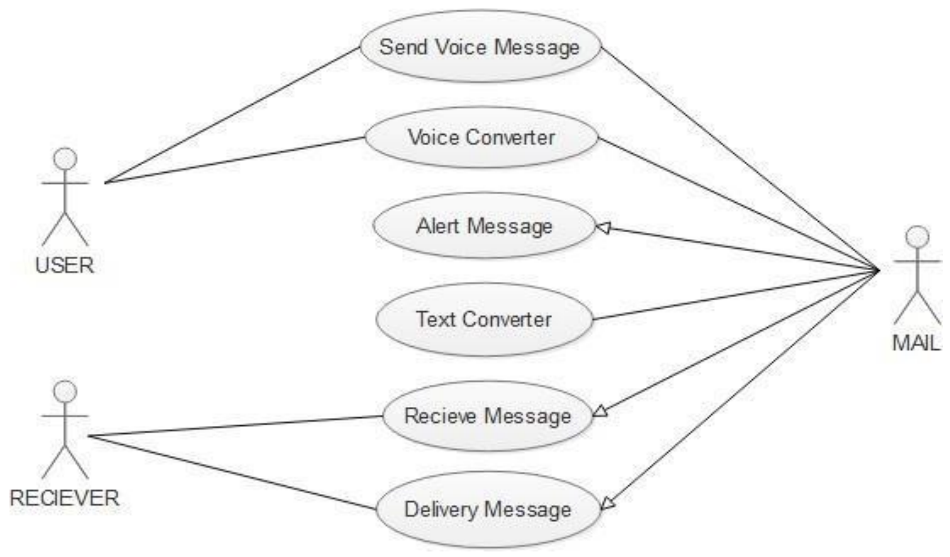


Fig 5.4; Use case diagram

A use-case diagram fig 5.4 visually represents the interactions between users (actors) and a system. In this voice-based email system, three actors are involved: User, Mail, and Receiver

- **User**: Sends a voice message and receives an alert message confirming delivery.
- **Mail**: Converts voice messages to text, processes and delivers the text message and confirms delivery to the user.
- **Receiver**: Receives the converted message and receives delivery confirmation.

In this use-case diagram, the **User** sends a voice message which is then converted into text by the **Mail** system. The system alerts the user upon delivery. The **Receiver** gets the converted message and receives a delivery confirmation. This diagram effectively showcases the interaction and processes involved in sending a voice-based email, ensuring successful communication and confirmation at each stage.

6. IMPLEMENTATION

The project "Voice-Based Email System for the Blind" focuses on creating an accessible email application tailored to visually impaired users, utilizing voice commands for all interactions. This system leverages speech recognition and text-to-speech technologies to facilitate seamless email communication. Key implementation components include a robust speech recognition engine, a natural language processing (NLP) module, and a user-friendly interface designed for audio interaction.

The system employs speech recognition technology, utilizing libraries like Google's Speech-to-Text API or CMU Sphinx, to convert spoken commands into text. This text is then processed by a natural language processing (NLP) module to interpret user intent. For instance, the command "Read my emails" triggers email retrieval from the user's inbox. The retrieved emails are then converted into speech using text-to-speech technology, enabling the user to listen to their email content.

The system supports essential email tasks such as composing, reading, deleting, and organizing emails. When composing an email, users dictate the content, and the system transcribes it into text. Recipients and subjects are specified through voice commands, with the system providing feedback to confirm accuracy before sending. For reading emails, the system reads out sender information, subject lines, and email bodies. Users can navigate their inbox, open specific emails, and perform actions like replying or deleting using voice commands.

Python, with its rich libraries for speech recognition (like SpeechRecognition) and text-to-speech (like gTTS), is a suitable language for system development. Integration with email services can be achieved using libraries like smtplib for sending emails and imaplib for retrieving emails. Prioritizing user security and privacy is crucial. Encryption and secure authentication methods should be implemented to safeguard personal email data.

6.1 SNIPPET CODE

Login.py

```
from tkinter import Message, Text, filedialog
import shutil
import tkinter as tk
import csv
import numpy as np
from PIL import Image, ImageTk
import tkinter.messagebox as tm
import MySQLdb
import cv2
import os
import pandas as pd
from gtts import gTTS
from playsound import playsound
import speech_recognition as sr
import random
import time
from datetime import datetime
import LoginPage as LP
import Start as s

uname = ""
password = ""
n = 0

mydb = MySQLdb.connect(host='localhost', user='root',
passwd='root', db='vmail')
conn = mydb.cursor()
cam = cv2.VideoCapture(0)

def pressed2(event):
    print(f'Button-2 pressed at x = {event.x}, y = {event.y}')

def TrackImages(dbuid):
    recognizer = cv2.face.LBPHFaceRecognizer_create()
    recognizer.read("TrainingImageLabel/Trainner.yml")
    harcascadePath = "haarcascade_frontalface_default.xml"
    faceCascade = cv2.CascadeClassifier(harcascadePath)
    df = pd.read_csv("StudentDetails/StudentDetails.csv")
    font = cv2.FONT_HERSHEY_SIMPLEX
    col_names = ['Id', 'Name', 'Date', 'Time']
    attendance = pd.DataFrame(columns=col_names)
    names = " ".join(df['Name'])
    attendance1 = pd.DataFrame(columns=['name'])

    while True:
        ret, im = cam.read()
        gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
        faces = faceCascade.detectMultiScale(gray, 1.2, 5)

        for (x, y, w, h) in faces:
            cv2.rectangle(im, (x, y), (x + w, y + h), (225, 0, 
0), 2)
            Id, conf = recognizer.predict(gray[y:y+h, x:x+w])
            print("id=", Id)
            print("conf=", conf)
            if conf < 60:
                print("outside if")
                tt = str(Id)
                print("dbuid=", dbuid)
                if dbuid != tt:
                    print("Inside if")
                    return False
                else:
                    print("inside else")
                    return True
            else:
                Id = 'Unknown'
                tt = str(Id)
                return False

            break
        cv2.putText(im, str(tt), (x, y + h), font, 1, (255, 255, 
255), 2)

        cv2.imshow('im', im)
        if cv2.waitKey(1) == ord('q'):
            cam.release()
            cv2.destroyAllWindows()
            break
```

```
def pressed3(event):
    global n
    print(f'Button-3 pressed at x = {event.x}, y = {event.y}')
    print("username", uname)
    print("password", password)

    cmd = f"SELECT uid, uname, pass FROM login WHERE
uname='{uname}' and pass='{password}'"
    print(cmd)
    conn.execute(cmd)
    cursor = conn.fetchall()
    uid = ""
    isRecordExist = 0
    for row in cursor:
        uid = row[0]
        isRecordExist = 1

    if isRecordExist == 1:
        if TrackImages(uid):
            cam.release()
            cv2.destroyAllWindows()
            tts = gTTS(text="Login Successfully", lang='en')
            ttsname = f"name{random.randint(0, 999)}.mp3"
            tts.save(ttsname)
            playsound(ttsname)
            os.remove(ttsname)
            n = 1
        else:
            tts = gTTS(text="Face is not matched", lang='en')
            ttsname = f"name{random.randint(0, 999)}.mp3"
            tts.save(ttsname)
            playsound(ttsname)
            os.remove(ttsname)
    else:
        tts = gTTS(text="Check Username and Password",
lang='en')
        ttsname = f"name{random.randint(0, 999)}.mp3"
        tts.save(ttsname)
        playsound(ttsname)
        os.remove(ttsname)

def process():
    global uname, password, n
    bgcolor = "#298a96"
    fgcolor = "white"
    window = tk.Tk()
    window.title("Voice based Email System")
    window.geometry('1000x600')
    window.configure(background=bgcolor)
    window.grid_rowconfigure(0, weight=1)
    window.grid_columnconfigure(0, weight=1)
    window.bind('<Button-2>', pressed2)
    window.bind('<Button-3>', pressed3)

    message1 = tk.Label(window, text="Voice based Email System",
bg=bgcolor, fg=fgcolor, width=50, height=3, font=('Helvetica',
40, 'bold'))
    message1.place(relx=0.5, rely=0.1, anchor='center')

    lbl = tk.Label(window, text="User Name", width=20, height=2,
fg=fgcolor, bg=bgcolor, font=('times', 20, 'bold'))
    lbl.place(x=150, y=150)
    txt = tk.Entry(window, width=20, bg="white", fg="black",
font=('times', 20, 'bold'))
    txt.place(x=450, y=165)

    lbl1 = tk.Label(window, text="Password", width=20, height=2,
fg=fgcolor, bg=bgcolor, font=('times', 20, 'bold'))
    lbl1.place(x=150, y=250)
    txt1 = tk.Entry(window, width=20, bg="white", fg="black",
font=('times', 20, 'bold'))
    txt1.place(x=450, y=265)

    sym, sym1 = LP.process()
    txt.insert('end', sym)
    txt1.insert('end', sym1)
    uname = txt.get()
    password = txt1.get()
    window.update()
    while True:
        time.sleep(1)
        uname = txt.get()
        password = txt1.get()
        print(uname)
        print(password)
        print("n=", n)
        if n == 1:
            window.destroy()
            s.process(uname)
            break
        window.update()

    window.mainloop()
```

main.py

```
 from gtts import gTTS
from playsound import playsound
import speech_recognition as sr
import random
import os
import SignUp as s
import Login as l


def process():
    tts = gTTS(text="Welcome to
the Voice-Based Email System",
lang='en')
    ran = random.randint(0, 999)
    ttsname = f"name{ran}.mp3"
    tts.save(ttsname)
    playsound(ttsname)
    os.remove(ttsname)
    print("Login")
    tts = gTTS(text="Say Login",
lang='en')
    ran = random.randint(0, 999)
    ttsname = f"hello1{ran}.mp3"
    tts.save(ttsname)
    playsound(ttsname)
    os.remove(ttsname)

    tts = gTTS(text="Your
choice", lang='en')
    ran = random.randint(0, 999)
    ttsname = f"hello2{ran}.mp3"
    tts.save(ttsname)
    playsound(ttsname)
    os.remove(ttsname)

    text = ""
    while text == "":
        # Voice recognition part
        recognizer =
sr.Recognizer()
        microphone =
sr.Microphone()

        with microphone as
source:

recognizer.adjust_for_ambient_no
ise(source)
            print("Your
choice:")
```

```
            audio =
recognizer.listen(source)
            print("ok done!!")

        try:
            text =
recognizer.recognize_google(audi
o)
            print(f"You said:
{text}")
        except
sr.UnknownValueError:
            print("Google Speech
Recognition could not understand
audio.")
        except sr.RequestError
as e:
            print(f"Could not
request results from Google
Speech Recognition service;
{e}")

        if text == "":
            tts =
gTTS(text="Error in message.
Please give input again",
lang='en')
            ran =
random.randint(0, 999)
            ttsname =
f"err{ran}.mp3"
            tts.save(ttsname)
            playsound(ttsname)
            os.remove(ttsname)

    if text.lower() in ['1',
'one', 'o n e', 'login',
'signin']:
        print("Login")
        l.process()

process()
```

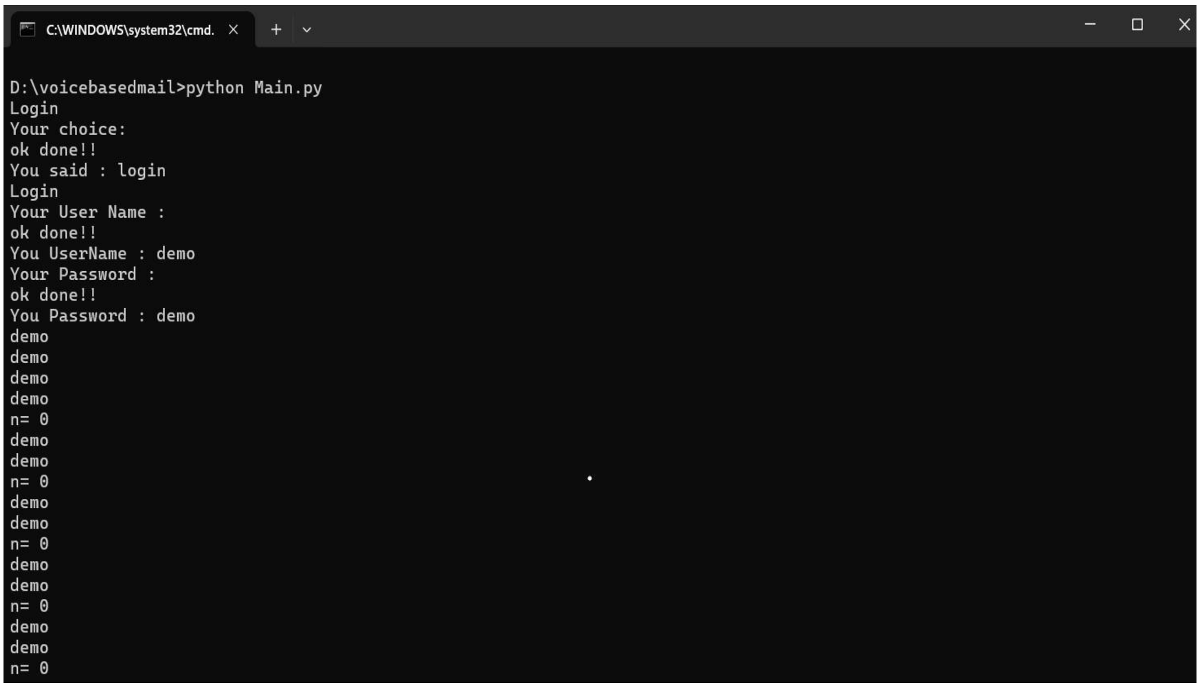
sendmaila.py

```
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.base import MIMEBase
from email import encoders
from playsound import playsound
import speech_recognition as sr
import random
import MySQLdb
import tkinter as tk
import time
import Start as s
from gtts import gTTS
import os

remail = ""
subject = ""
bom = ""
sender_email = ""
password = ""
fname = ""
n = 0
```

```
mydb = MySQLdb.connect(host='localhost', user='root',
passwd='root', db='vmail')
conn = mydb.cursor()


def pressed2(event):
    print(f'Button-2 pressed at x = {event.x}, y = {event.y}')


def pressed3(event):
    global n
    print(f'Button-3 pressed at x = {event.x}, y = {event.y}')
    n = 1
    fromaddr = sender_email
    toaddr = remail
    msg = MIMEMultipart()
    msg['From'] = fromaddr
    msg['To'] = toaddr
    msg['Subject'] = subject
    body = bom
    msg.attach(MIMEText(body, 'plain'))

    filename = fname
    attachment = open(fname, "rb")
    p = MIMEBase('application', 'octet-stream')
    p.set_payload(attachment.read())
    encoders.encode_base64(p)
    p.add_header('Content-Disposition', f"attachment; filename=
{filename}")
    msg.attach(p)

    s = smtplib.SMTP('smtp.gmail.com', 587)
    s.starttls()
    s.login(fromaddr, password)
    text = msg.as_string()
    s.sendmail(fromaddr, toaddr, text)
    s.quit()
```

```
def display(uname):
    global remail, subject, bom, sender_email, password, n, fname
    bgcolor = "#298a96"
    fgcolor = "white"
    window = tk.Tk()
    window.title("Voice-Based Email System")
    window.geometry('1280x720')
    window.configure(background=bgcolor)
    window.grid_rowconfigure(0, weight=1)
    window.grid_columnconfigure(0, weight=1)
    window.bind('<Button-2>', pressed2)
    window.bind('<Button-3>', pressed3)

    message1 = tk.Label(window, text="Voice-Based Email System",
bg=bgcolor, fg=fgcolor, width=50, height=3, font=('times', 30,
'bold'))
    message1.place(relx=0.5, rely=0.1, anchor='center')

    lbl = tk.Label(window, text="Receiver Email", width=20,
height=2, fg=fgcolor, bg=bgcolor, font=('times', 15, 'bold'))
    lbl.place(x=300, y=150)
    txt = tk.Entry(window, width=40, bg="white", fg="black",
font=('times', 15, 'bold'))
    txt.place(x=600, y=160)

    lbl1 = tk.Label(window, text="Subject", width=20, height=2,
fg=fgcolor, bg=bgcolor, font=('times', 15, 'bold'))
    lbl1.place(x=300, y=250)
    txt1 = tk.Entry(window, width=40, bg="white", fg="black",
font=('times', 15, 'bold'))
    txt1.place(x=600, y=260)

    lbl2 = tk.Label(window, text="Message", width=20, height=2,
fg=fgcolor, bg=bgcolor, font=('times', 15, 'bold'))
    lbl2.place(x=300, y=350)
    txt2 = tk.Entry(window, width=40, bg="white", fg="black",
font=('times', 15, 'bold'))
    txt2.place(x=600, y=360)

    lbl3 = tk.Label(window, text="Filename", width=20, height=2,
fg=fgcolor, bg=bgcolor, font=('times', 15, 'bold'))
    lbl3.place(x=300, y=450)
    txt3 = tk.Entry(window, width=40, bg="white", fg="black",
font=('times', 15, 'bold'))
    txt3.place(x=600, y=460)

    print("remail in display", remail)

    txt.insert('end', remail)
    txt1.insert('end', subject)
    txt2.insert('end', bom)
    txt3.insert('end', fname)

    window.update()
    while True:
        time.sleep(1)
        remail = txt.get()
        subject = txt1.get()
        bom = txt2.get()
        fname = txt3.get()
        print(remail)
        print(subject)
        print(bom)
        print(n)
        if n == 1:
            window.destroy()
            s.process(uname)
            break
        window.update()
    window.mainloop()


def process1(sym):
    global remail, subject, bom, sender_email, password, fname
    cmd = f"SELECT email, epasssword FROM login WHERE
uname='{sym}'"
    print(cmd)
    conn.execute(cmd)
    cursor = conn.fetchall()
    sender_email = ""
    password = ""
    for row in cursor:
        print(row[0])
        sender_email = row[0]
        password = row[1]

    receiver_email = "dmindsoftblore@gmail.com"
    r = sr.Recognizer()
    m = sr.Microphone()
    with m as source:
        r.adjust_for_ambient_noise(source)

    tts = gTTS(text="Enter Receiver Email.", lang='en')
    ttsname = "sub1.mp3"
    tts.save(ttsname)
    playsound(ttsname)
    os.remove(ttsname)

    receiver_email1 = ""
    while receiver_email1 == "":
        with sr.Microphone() as source:
            print("Your Receiver Email:")
            audio = r.listen(source)
            print("ok done!!")
        try:
            receiver_email1 = r.recognize_google(audio)
            print("Your Receiver Email:", receiver_email1)
        except sr.UnknownValueError:
            print("Google Speech Recognition could not understand 
audio.")
        except sr.RequestError as e:
            print(f"Could not request results from Google Speech 
Recognition service; {e}")
        if receiver_email1 == "":
            tts = gTTS(text="Error in Receiver Email. Please Give 
Input Again", lang='en')
            ran = random.randint(0, 999)
            ttsname = f"err{ran}.mp3"
            tts.save(ttsname)
            playsound(ttsname)
            os.remove(ttsname)

    words = receiver_email1.split()
    modified_mail = "".join(['_' if word == 'underscore' else '.' 
if word == 'dot' else '@' if word == 'at' else word for word in 
words])
    receiver_email = modified_mail

    sub = ""
    msg = ""
    fname = ""

    while sub == "":
        print(receiver_email)
        tts = gTTS(text="Your Subject.", lang='en')
        ttsname = "sub.mp3"
        tts.save(ttsname)
        playsound(ttsname)
        os.remove(ttsname)
        with sr.Microphone() as source:
            print("Your Subject:")
            audio = r.listen(source)
            print("ok done!!")
        try:
            sub1 = r.recognize_google(audio)
            print("Your Subject:", sub1)
            sub = sub1
        except sr.UnknownValueError:
            print("Google Speech Recognition could not understand
audio.")
        except sr.RequestError as e:
            print(f"Could not request results from Google Speech
Recognition service; {e}")
        if sub == "":
            tts = gTTS(text="Error in Subject. Please Give Input
Again", lang='en')
            ran = random.randint(0, 999)
            ttsname = f"err{ran}.mp3"
            tts.save(ttsname)
            playsound(ttsname)
            os.remove(ttsname)

    while msg == "":
        tts = gTTS(text="Body of the Message.", lang='en')
        ran = random.randint(0, 999)
        ttsname = f"sub{ran}.mp3"
        tts.save(ttsname)
        playsound(ttsname)
        os.remove(ttsname)

        with sr.Microphone() as source:
            print("Body of the Message:")
            audio = r.listen(source)
            print("ok done!!")
        try:
            text1 = r.recognize_google(audio)
            print("You said:", text1)
            msg = text1
        except sr.UnknownValueError:
            print("Google Speech Recognition could not understand 
audio.")
        except sr.RequestError as e:
            print(f"Could not request results from Google Speech 
Recognition service; {e}")
        if msg == "":
            tts = gTTS(text="Error in Body of Message. Please 
Give Input Again", lang='en')
            ran = random.randint(0, 999)
            ttsname = f"err{ran}.mp3"
            tts.save(ttsname)
            playsound(ttsname)
            os.remove(ttsname)

    while fname == "":
        tts = gTTS(text="File name with extension", lang='
```

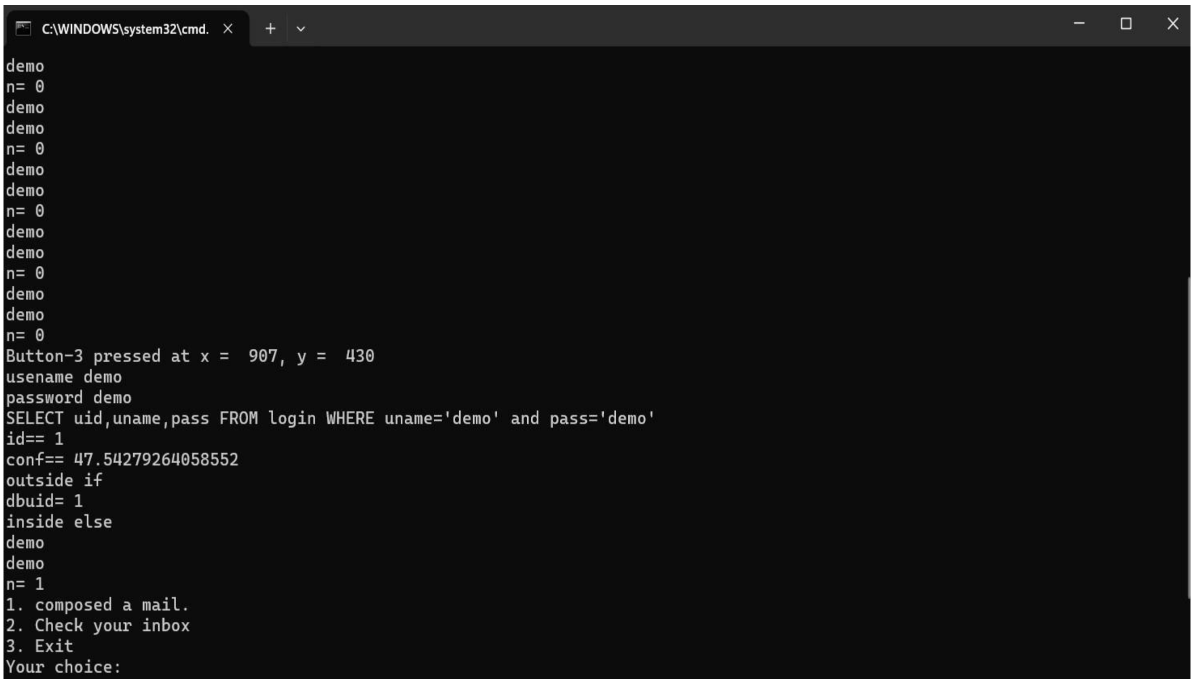
6.2 SCREENSHOTS


C:\WINDOWS\system32\cmd.

```
D:\voicebasedmail>python Main.py
Login
Your choice:
ok done!!
You said : login
Login
Your User Name :
ok done!!
You UserName : demo
Your Password :
ok done!!
You Password : demo
demo
demo
demo
demo
n= 0
demo
demo
n= 0
demo
demo
n= 0
demo
demo
n= 0
demo
demo
n= 0
```

Fig 6.2.1; User authentication

The above Screenshot describes the user authentication process done by the AI to login the user into the system.


C:\WINDOWS\system32\cmd.

```
demo
n= 0
demo
demo
n= 0
demo
demo
n= 0
demo
demo
n= 0
demo
demo
n= 0
Button-3 pressed at x =  907, y =  430
usename demo
password demo
SELECT uid,uname,pass FROM login WHERE uname='demo' and pass='demo'
id== 1
conf== 47.54279264058552
outside if
dbuid= 1
inside else
demo
demo
n= 1
1. composed a mail.
2. Check your inbox
3. Exit
Your choice:
```

Fig 6.2.2; Confirming User loaded Credentials

The above screenshot shows that the AI is giving choices to user for how to send the mail.

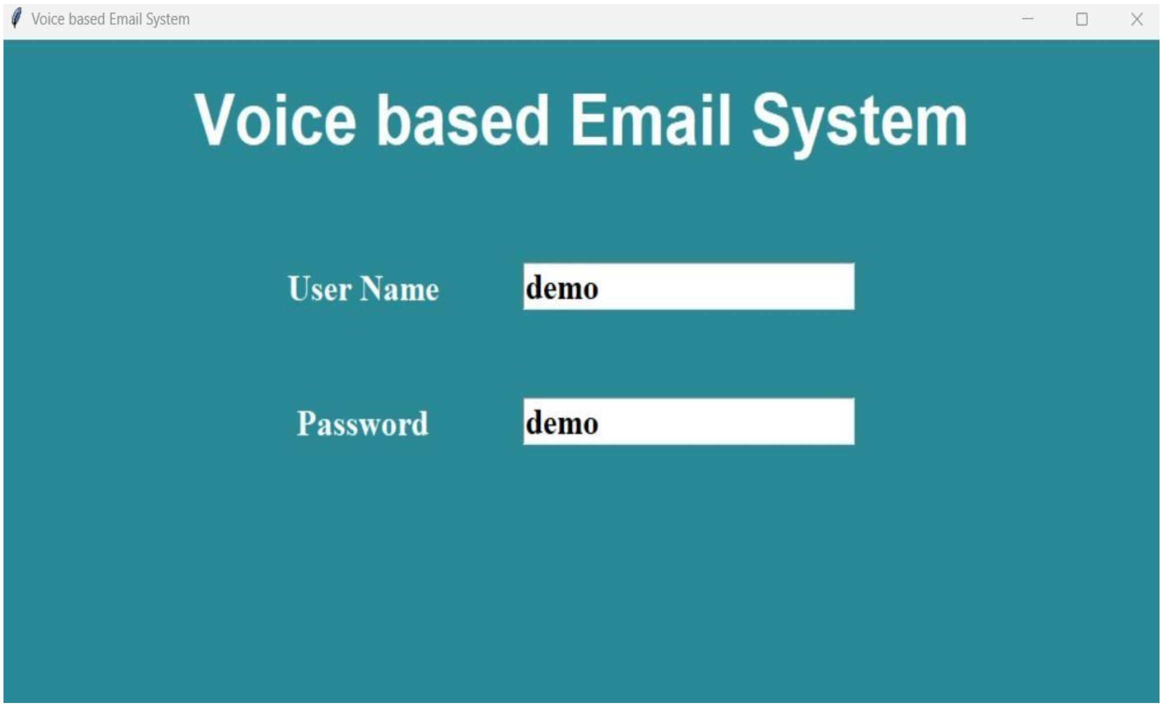
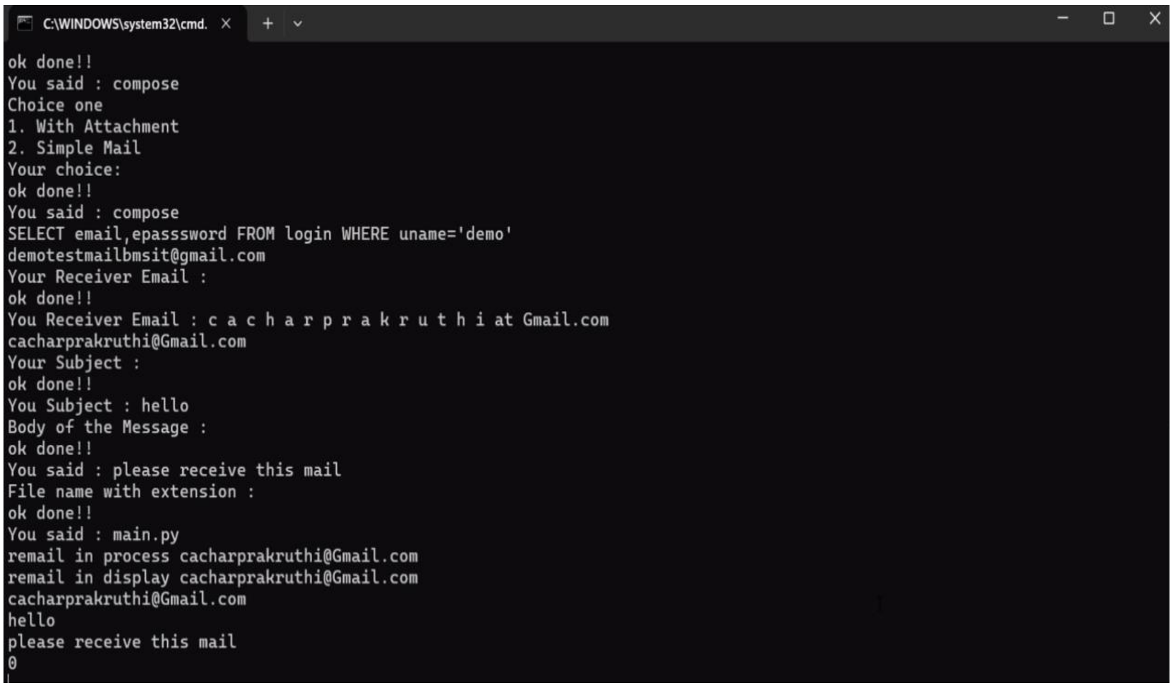


Fig 6.2.3; Login Credentials with Face recognition

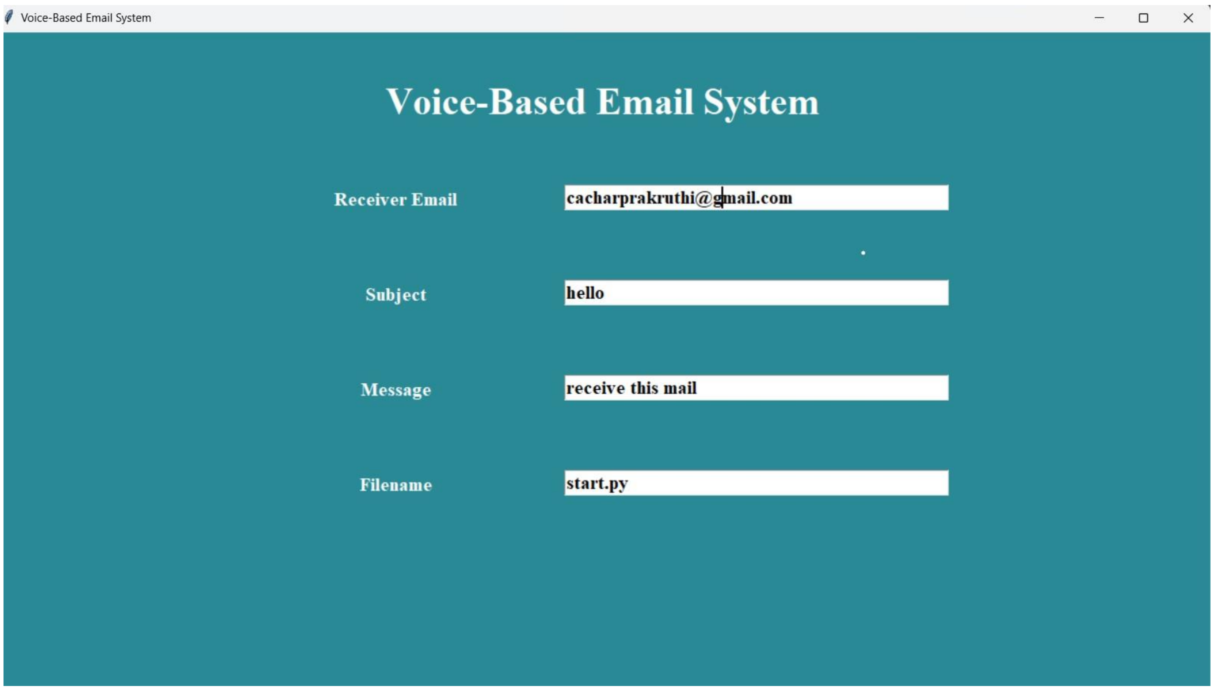
This Screenshot represents the login info loaded by user through speech and as user does right click, face recognition takes incharge for authentication.



```
ok done!!
You said : compose
Choice one
1. With Attachment
2. Simple Mail
Your choice:
ok done!!
You said : compose
SELECT email,epasssword FROM login WHERE uname='demo'
demotestmailbmsit@gmail.com
Your Receiver Email :
ok done!!
You Receiver Email : c a c h a r p r a k r u t h i at Gmail.com
cacharprakruthi@Gmail.com
Your Subject :
ok done!!
You Subject : hello
Body of the Message :
ok done!!
You said : please receive this mail
File name with extension :
ok done!!
You said : main.py
remail in process cacharprakruthi@Gmail.com
remail in display cacharprakruthi@Gmail.com
cacharprakruthi@Gmail.com
hello
please receive this mail
0
```

Fig 6.2.4; Sending mail with attachment

The above screenshot describes conversation between user and AI for sending a mail with Attachment.


Voice-Based Email System

Voice-Based Email System

Receiver Email: cacharprakruthi@gmail.com

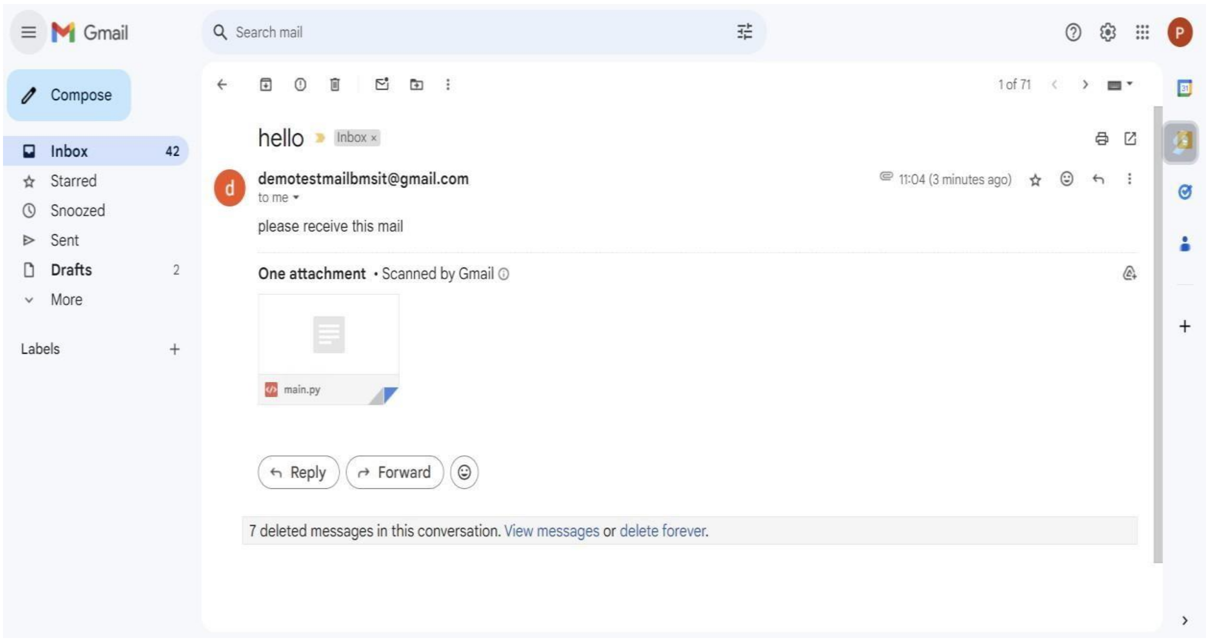
Subject: hello

Message: receive this mail

Filename: start.py

Fig 6.2.5; Confirmation of Email details

This Screenshot represents the Email info loaded by user through speech and as user does right click, AI confirms that the mail is sent with an attachment file.



hello Inbox ×

demotestmailbmsit@gmail.com
to me
11:04 (3 minutes ago)

please receive this mail

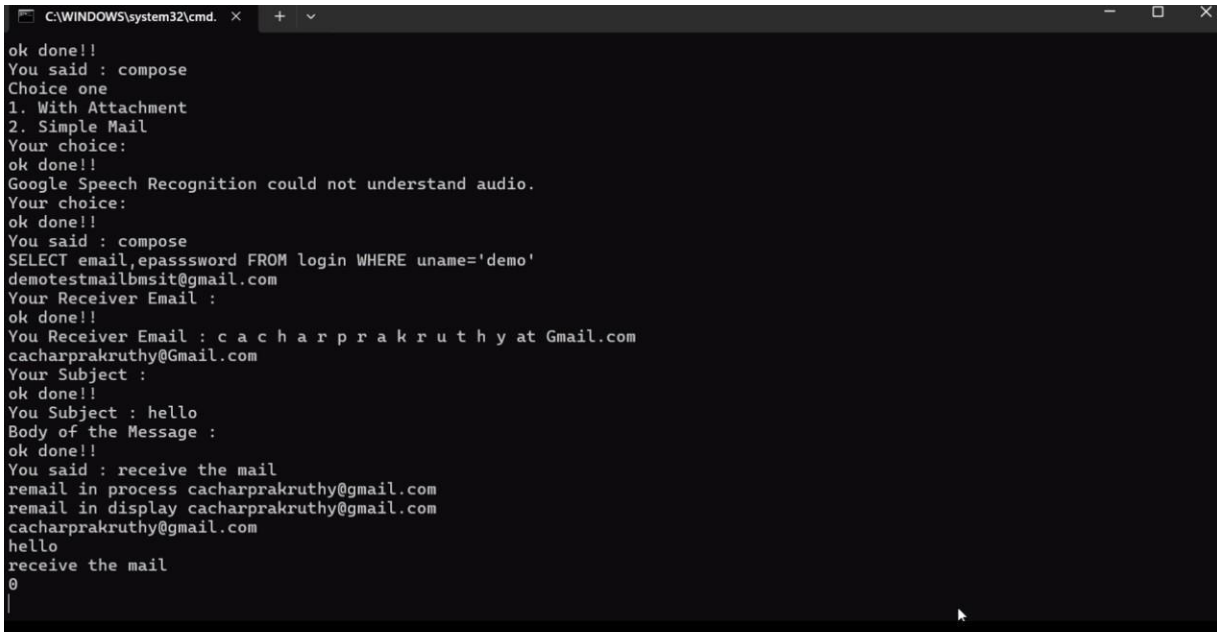
One attachment • Scanned by Gmail

main.py

7 deleted messages in this conversation. View messages or delete forever.

Fig 6.2.6; Recipient’s Inbox

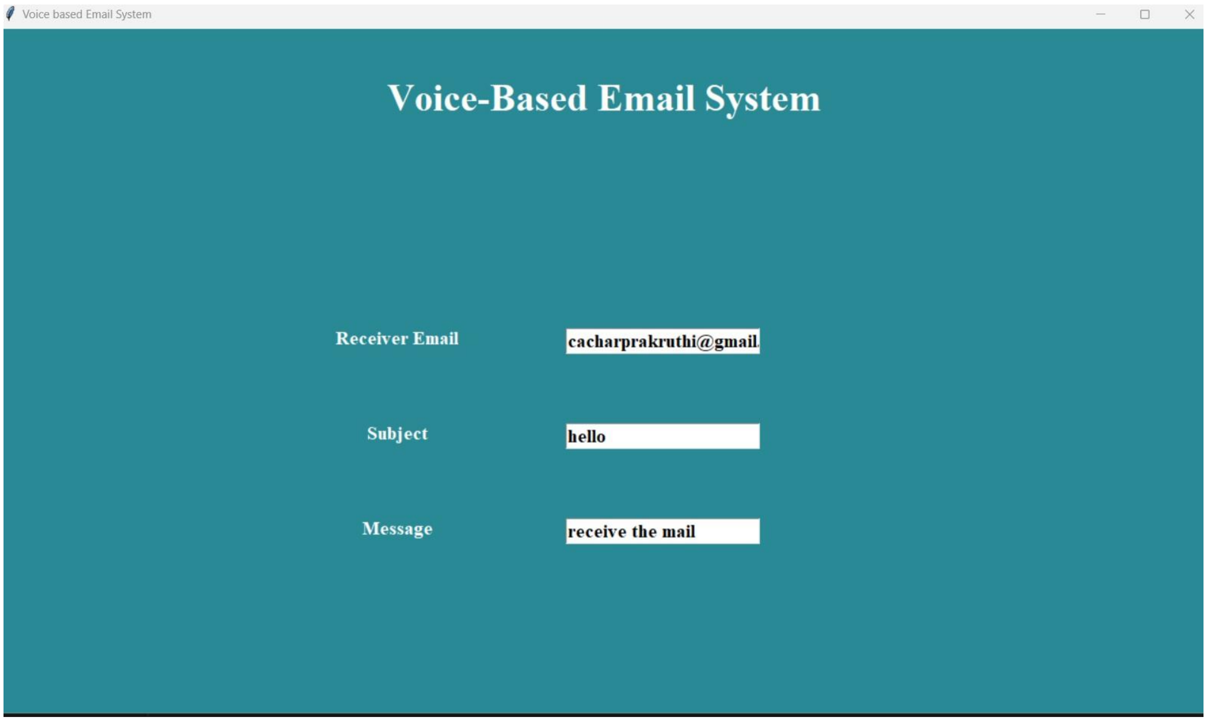
This shows the actual Email received by a recipient to whom the user has mailed with the help of AI in Gmail inbox.



```
C:\WINDOWS\system32\cmd.
ok done!!
You said : compose
Choice one
1. With Attachment
2. Simple Mail
Your choice:
ok done!!
Google Speech Recognition could not understand audio.
Your choice:
ok done!!
You said : compose
SELECT email,epasssword FROM login WHERE uname='demo'
demotestmailbmsit@gmail.com
Your Receiver Email :
ok done!!
You Receiver Email : c a c h a r p r a k r u t h y at Gmail.com
cacharprakruthy@Gmail.com
Your Subject :
ok done!!
You Subject : hello
Body of the Message :
ok done!!
You said : receive the mail
remail in process cacharprakruthy@gmail.com
remail in display cacharprakruthy@gmail.com
cacharprakruthy@gmail.com
hello
receive the mail
0
```

Fig 6.2.7; Sending simple mail

The above screenshot describes conversation between user and AI for sending a mail with without attachment or Simple mail.



Voice based Email System

Voice-Based Email System

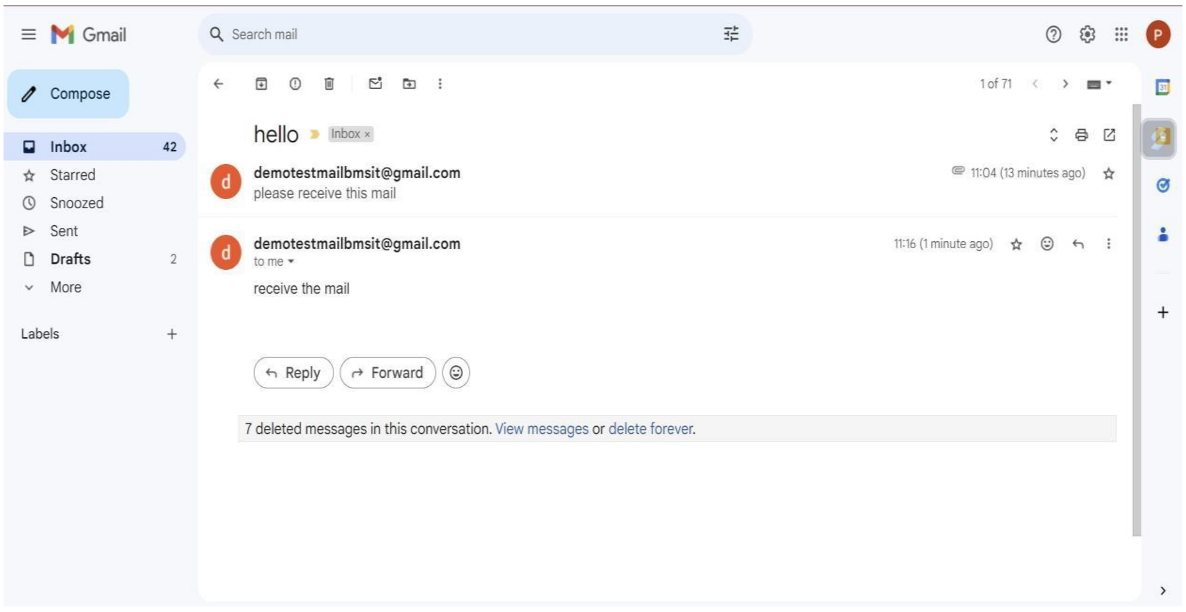
Receiver Email: cacharprakruthi@gmail

Subject: hello

Message: receive the mail

Fig 6.2.8; Confirmation of Email details

This Screenshot represents the E-mail info loaded by user through speech and as user does right click, AI confirms that the mail is sent in the form of simple mail with a message.



hello Inbox ×

demotestmailbmsit@gmail.com 11:04 (13 minutes ago)
please receive this mail

demotestmailbmsit@gmail.com 11:16 (1 minute ago)
to me
receive the mail

7 deleted messages in this conversation. View messages or delete forever.

Fig 6.2.9; Recipient's Inbox

This shows the actual E-mail received by a recipient to whom the user has mailed with the help of AI in Gmail inbox.

7. SOFTWARE TESTING

Effective software testing is crucial for ensuring the functionality and reliability of a voice-based email system for blind users. The testing process involves various strategies to identify and fix issues, ensuring the application performs as intended.

7.1 UNIT TESTING

- Unit testing is a crucial phase in software development aimed at verifying the functionality of individual components or units of the system. Each module, such as voice recognition, text-to-speech conversion, email management, and user authentication, undergoes testing in isolation to ensure it performs as expected. The primary objective is to detect and address defects early in the development cycle, thereby improving the overall quality and reliability of the software.
- **Voice Recognition Module**: the voice recognition module plays a pivotal role. It must accurately interpret spoken commands and convert them into actionable tasks such as composing, reading, or managing emails. This module involve feeding it with various voice commands, testing different accents and speech patterns to ensure robustness and accuracy in speech-to-text conversion.
- **Text-to-Speech Conversion**: Another critical component is the text-to-speech module, responsible for converting email content into spoken words. Unit testing here focuses on verifying that text is converted fluently and understandably, ensuring that the output maintains clarity and coherence. Tests may include varying text lengths, complex sentence structures, and special characters to validate comprehensive functionality.
- **Email Management**: This module handles the core functionalities of the email system, including sending, receiving, reading, and deleting emails through voice commands. Unit tests are designed to simulate these operations and verify their correctness. For instance, sending an email should accurately deliver the message to the intended recipient, while reading an email should correctly articulate its content to the user.
- **User Authentication**: Security and user authentication are paramount in any application. Unit tests for authentication mechanisms ensure that the login and logout processes work securely and efficiently. This involves testing scenarios with valid and invalid credentials to confirm that access is granted or denied appropriately, maintaining the integrity of user accounts and data.

7.2 AUTOMATION TESTING

Automation testing complements unit testing by leveraging automated tools to execute test cases efficiently. This method is particularly beneficial for repetitive tasks and regression testing, ensuring that new code changes do not introduce unintended issues or regressions in existing functionality. The Automation testing can validate critical aspects such as:

- **Accuracy of Voice Recognition**: Automated tests can simulate a variety of voice inputs to verify that the recognition module consistently interprets commands correctly across different scenarios.
- **Quality of Text-to-Speech Conversion**: By automating tests for text-to-speech conversion, developers can ensure that the system consistently delivers clear and intelligible speech outputs without errors or distortions.
- **Functionalities of Email Operations**: Automation can be used to test the end-to-end functionalities of email operations, including sending, receiving, and managing emails through voice commands. This approach allows testers to cover a wide range of scenarios efficiently, ensuring robust performance.
- **Accessibility Features**: Automated tests can validate accessibility features specific to blind users, such as the system's ability to provide meaningful auditory feedback and handle errors gracefully. This ensures that the application remains fully accessible and user-friendly in real-world usage scenarios.

7.3 TEST CASES

Developing comprehensive test cases is essential to validate the functionalities and performance of a voice-based email system for blind users. Test cases are detailed scenarios that outline specific inputs, execution steps, and expected outcomes across different modules and user interactions. Key test cases   for such a system are as following table 7.3.1

Table 7.3.1; Test Cases

Test Case ID	**Module**	**Test Case Description**	**Expected Outcome**	**Test Data**
TC1	Voice Recognition	Verify recognition of common email commands (e.g., 'Read email,' 'Compose email')	Commands are accurately recognized and processed.	Examples: 'Read email,' 'Compose email,' 'Reply to email'
TC2	Voice Recognition	Test command recognition with various accents and speech patterns	Commands are consistently recognized regardless of accent or speech pattern.	Examples: Commands spoken with American, British, and Indian accents. Commands spoken with slow and fast speech rates.
TC3	Text-to-Speech Conversion	Verify clarity and intelligibility of speech output for emails of varying lengths and complexities	Email content is converted into clear and understandable speech without errors or omissions.	Examples: Short emails, long emails with attachments, emails with complex formatting
TC4	Email Operations	Test sending and receiving emails using voice commands	Emails are delivered accurately and promptly.	Examples: Sending emails to different recipients, receiving emails with attachments

TC5	Email Operations	Test reading and managing emails (e.g., deleting) using voice commands	Email content is correctly read aloud, and users can effectively manage their inbox.	Examples: Reading specific emails by subject or sender, deleting emails
TC6	User Authentication	Test login and logout processes using voicebased authentication	Users can securely log in and out with valid credentials. Invalid credentials are handled appropriately.	Examples: Login with valid username and password, Login with invalid username or password
TC7	Accessibility	Verify provision of auditory feedback and error handling for blind users	System provides meaningful auditory feedback and guides users through errors gracefully.	Examples: System announces new emails, System provides audio cues for error messages

In conclusion, unit testing, automation testing, and comprehensive test case development are integral to ensuring the functionality, reliability, and accessibility of a voice-based email system tailored for blind users. By rigorously testing each module and scenario, developers can identify and resolve issues early in the development lifecycle, ultimately delivering a robust and user-friendly application that meets the specific needs of visually impaired individuals. This approach not only enhances the quality of the software but also fosters confidence in its performance under real-world usage condition.

8. CONCLUSION

The development journey of the voice-based email system for blind users has been guided by a commitment to innovation and accessibility, as discussed across our interactions. From initial planning to implementation and testing, every phase has underscored the importance of addressing the specific needs of visually impaired individuals with cutting-edge technology.

Throughout our discussions, we emphasized the significance of unit testing, automation testing, and detailed test case development to ensure the system's functionality and reliability. By meticulously testing components such as voice recognition, text-to-speech conversion, email management, and user authentication, we aimed to deliver a seamless user experience. These efforts not only validated the system's performance but also highlighted areas for refinement and enhancement. Automation testing, as suggested, played a pivotal role in validating the system's robustness across various scenarios. By automating repetitive tests and regression testing, we ensured that updates and modifications did not compromise existing functionalities. This approach not only accelerated the testing process but also strengthened our confidence in the system's ability to meet the diverse needs of blind users reliably. Our exploration of comprehensive test cases encompassed diverse user interactions, from basic command recognition to complex email operations and secure authentication processes. Each test case served to validate functionality, accessibility features, and user interface responsiveness, aligning closely with the project's overarching goal of inclusivity and usability. Looking ahead, the voice-based email system for blind users stands poised to make a significant impact in fostering communication independence. The ongoing user feedback and iterative improvements, ensuring that the system evolves in tandem with technological advancements and user expectations. By automating repetitive tests and regression testing, we ensured that updates and modifications did not compromise existing functionalities. This approach not only accelerated the testing process but also strengthened our confidence in the system's ability to meet the diverse needs of blind users reliably. Our exploration of comprehensive test cases encompassed diverse user interactions, from basic command recognition to complex email operations and secure authentication processes. Each test case served to validate functionality, accessibility features, and user interface responsiveness, aligning closely with the project's overarching goal of inclusivity and usability.

Hence, this project has exemplified our commitment to leveraging technology for social good, empowering visually impaired individuals with techniques that improve their daily lives. By integrating advanced testing methodologies and user-centered design principles, we have laid a foundation for a robust, reliable, and user-friendly application. This journey reflects our dedication to innovation, accessibility, and inclusivity, paving the way for future advancements in accessible technology.

9. FUTURE ENHANCEMENTS

Moving forward, several enhancements can be considered to further improve the functionality, accessibility, and user experience of the voice-based email system for blind users:

- **Natural Language Processing (NLP) Integration:** Integrate advanced NLP algorithms to enhance the system's ability to understand and respond to natural language commands more accurately. This could involve parsing more complex instructions and providing more intuitive responses.
- **Personalization and Customization Options:** Implement features that allow users to customize voice commands, email preferences, and interface settings according to their individual needs and preferences.
- **Enhanced Security Features:** Strengthen security measures by implementing additional authentication methods such as biometric recognition (e.g., voice biometrics) to enhance user privacy and protect sensitive email content.
- **Integration with Assistive Technologies:** Explore integration with other assistive technologies and devices commonly used by visually impaired individuals, such as screen readers or Braille displays, to provide a seamless and integrated user experience across different platforms.
- **Multi-Language Support:** Extend support for multiple languages and dialects to cater to a diverse user base globally. This enhancement would involve expanding the language models used for both voice recognition and text-to-speech conversion.
- **Offline Functionality:** Develop offline capabilities that allow users to perform basic email operations (e.g., composing, reading, deleting) without requiring a constant internet connection.
- **Continuous User Feedback and Iterative Improvements:** Establish mechanisms for collecting user feedback and analytics to identify areas for improvement continuously.
- **Integration with Third-Party Services:** Explore partnerships and integrations with popular email service providers and productivity tools to offer seamless synchronization and expanded functionality.

- **Accessibility Audits and Compliance:** Conduct regular accessibility audits and to continually improve the user interface and interaction design for blind and visually impaired users.
- **AI-driven Smart Assistants:** Implement AI-driven smart assistants that can proactively assist users with email management tasks, such as summarizing long emails, prioritizing messages, or suggesting responses based on user preferences and historical interactions.

In conclusion, enhancing the voice-based email system with advanced NLP, personalization, stronger security, assistive technology integration, multi-language support, offline functionality, user feedback mechanisms, third-party service integration, accessibility audits, and AI-driven assistants will significantly improve functionality, accessibility, and user experience for blind users.

10. REFERENCES

[1] Screen Readers: Evolution and Accessibility Improvements" by Jones, M., & Petrie, H. (2017) - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8636846/

[2] Challenges in the Development of Assistive Technologies for the Visually Impaired" by Shinohara, K., & Wobbrock, J. O. (2016) - https://www.mdpi.com/1424-8220/24/11/3572

[3] Speech-to-Text and Text-to-Speech Technology for Accessibility" by Story, M. F., & Maxim, B. E. (2016) - https://ieeexplore.ieee.org/abstract/document/10212222

[4] Incorporating Voice Commands into Assistive Technologies for Improved Usability" by Cooper, M., & Edler, S. (2019) – https://link.springer.com/article/10.1007/s11528-024-009876

[5] Advances in Speech Recognition Technology for Assistive Applications" by Jiang, H., & Li, X. (2020) – https://dl.acm.org/doi/10.1016/j.eswa.2024.124159

[6] A Review of Text-to-Speech Synthesis from the Perspective of Assistive Technology" by Tao, J., & Tan, T. (2017) – https://jait.us/uploadfile/2022/0831/20220831054604906.pdf

[7] Enhancing Screen Reader Usability with Improved Speech Synthesis" by Alm, N., & Newell, A. F. (2018) - https://ieeexplore.ieee.org/document/10160209

[8] Voice-Based Email Systems: Improving Accessibility for the Visually Impaired" by Roy, P., & Samanta, D. (2019) - https://ijrpr.com/uploads/V5ISSUE4/IJRPR24786.pdf

[9] Speech Recognition Accuracy in Assistive Technologies: Evaluation and Improvements" by Liao, Y., & He, J. (2020) - https://www.tandfonline.com/doi/full/10.1080/10400435.2023.2260860

[10] Designing Voice Command Systems for Digital Accessibility" by Brooks, J., & Peters, G. (2018) - https://www.tandfonline.com/doi/full/10.1080/10400435.2021.1945705

[11] Project Idea | Voice Based Email for Visually Challenged - https://www.geeksforgeeks.org/project-idea-voice-based-email-visually-challenged/

[12] Playing and Recording Sound in Python- https://realpython.com/playing-and-recordingsound-python/