# Inclusivity in Action: The Visually Impaired Service Desk Agent for Appointment Booking

**Mr. Manikandan S[1], Dr. S. Deepa Jothi[2]**

[1]Student, [2]Associate Professor
[1, 2]Department of Computer Science, SRM Institute of Science and Technology

**Abstract**

**"Inclusivity in Action: The Visually Impaired Service Desk Agent for Appointment Booking is designed to empower visually impaired individuals to work as customer service executives handling appointment bookings. It introduces a voice-based interaction system integrated with transcription and a virtual assistant. By leveraging Whisper for transcription, pyttsx3 for text-to-speech, and a Flask backend with a React-based frontend, the application ensures seamless support handling. The project demonstrates that with suitable AI tools and thoughtful design, accessibility barriers in corporate environments can be reduced effectively.**

**The system facilitates end-to-end customer interaction without the need for visual interfaces, thereby placing blind users on par with sighted support agents. Call recordings are automatically transcribed, allowing the virtual assistant to respond contextually or assist the agent in navigating options. The interface is optimized for keyboard navigation and auditory feedback, ensuring full operational independence. Additionally, the solution is scalable and can be adapted to different domains like healthcare, education, and IT services.**

**By focusing on task automation and real-time support, the application not only enhances accessibility but also improves efficiency and customer satisfaction. Through this initiative, we aim to set a precedent for inclusive design in AI-powered service desk solutions. The project underscores the importance of equal employment opportunities and the role of assistive technologies in bridging systemic gaps.**

## LIST OF SYMBOLS AND ABBREVIATIONS

| Abbreviation | Description |
|---|---|
| AI | Artificial Intelligence |
| ML | Machine Learning |
| BSA | Blind Service Desk Agent |
| UI | User Interface |
| UX | User Experience |
| API | Application Programming Interface |
| NLP | Natural Language Processing |
| TTS | Text-to-Speech |
| STT | Speech-to-Text |

| | |
|---|---|
| JSON | JavaScript Object Notation |
| IDE | Integrated Development Environment |
| SQL | Structured Query Language |
| DB | Database |
| VS Code | Visual Studio Code |
| HTTP | HyperText Transfer Protocol |
| REST | Representational State Transfer |
| CPU | Central Processing Unit |
| RAM | Random Access Memory |
| JS | JavaScript |

# CHAPTER 1 INTRODUCTION

## 1.1. General

Over the past few decades, technological advancement has transformed the global work environment, particularly in the domain of customer service. However, a significant section of the population, particularly visually impaired individuals, remains underrepresented in such roles due to the heavy reliance on visual interfaces. According to the World Health Organization (WHO), at least 2.2 billion people globally have a vision impairment or blindness. Despite this large demographic, opportunities for them in traditional support centers are limited.

Customer service operations typically depend on visual cues: screen-based ticketing systems, web chats, form-based query logging, and dashboards for data retrieval. These systems often remain inaccessible or inefficient for blind individuals, thereby excluding them from roles they are otherwise fully capable of performing.

To bridge this gap, there is a growing need for assistive technologies tailored to customer support workflows. Voice-assisted interfaces, real-time transcription, and AI-based decision support can enable blind agents to handle customer interactions with autonomy. The integration of natural language processing and speech recognition models offers an inclusive path forward. Creating such accessible systems not only promotes equity but also taps into a skilled and underutilized workforce. By reimagining service desk environments through the lens of accessibility, we can make a substantial stride toward workplace inclusivity.

## 1.2. Problem Statement

The modern service desk environment is fast-paced and information-driven. Support agents are required to:

- Understand customer issues quickly
- Log tickets accurately
- Retrieve and update customer records
- Communicate resolution steps clearly

All these tasks are predominantly screen-based, which poses a major challenge for blind individuals. Existing screen readers can provide basic accessibility, but they are often slow, prone to errors, and

create cognitive overload due to their linear reading of screen contents.
Thus, there is a dire need for a new system: one that relies primarily on voice rather than sight. A system that can allow a visually impaired agent to:
•Hear transcriptions of customer calls
•Speak their responses
•Confirm appointment bookings without depending heavily on visual interfaces.
To address these challenges, we propose the development of a voice-centric service desk application designed specifically for blind support agents. This system will integrate advanced speech recognition, natural language processing, and text-to-speech technologies to provide a seamless and intuitive experience. Agents will be able to interact with customer queries, navigate knowledge bases, and log tickets entirely through voice commands. The system will also offer contextual voice prompts, reducing the need for memorization and manual navigation. In addition, automated summarization of conversations will help agents quickly grasp the issue at hand. The platform will support multi-language inputs and outputs to cater to global customers. Voice confirmations for actions such as ticket submission or escalation will enhance confidence and accuracy. Overall, the solution aims to empower blind individuals with the tools to perform on par with their sighted peers in a demanding support environment.

### 1.3. Importance of Accessibility

Accessibility in digital workspaces is no longer an optional feature but a human rights necessity. The United Nations Convention on the Rights of Persons with Disabilities (CRPD) emphasizes equal access to work opportunities, including the digital environment. Providing accessible technology solutions is vital for organizations to:
• Promote diversity and inclusion
• Comply with global accessibility laws (like ADA, Section 508)
• Leverage untapped talent pools
This project thus emerges as a critical innovation to bridge the accessibility gap in service desk operations.
Inclusive design benefits everyone—not just users with disabilities—by fostering more intuitive and user-friendly interfaces. Enabling visually impaired individuals to contribute meaningfully to the workforce also promotes economic independence and psychological well-being. Organizations embracing accessibility experience improved brand perception and social responsibility.
Moreover, as remote and hybrid work becomes the norm, accessible digital tools become even more essential. Failure to address accessibility can lead to legal risks, lost productivity, and reduced workforce morale. By integrating assistive technologies at the core of system design, we ensure long-term scalability, compliance, and innovation. Ultimately, true digital transformation must include accessibility as a fundamental principle.

### 1.4. Evolution of Assistive Technology

Historically, visually impaired individuals have relied on:
- Braille systems
- Screen readers (like JAWS, NVDA)
- Tactile keyboards

While these technologies have been instrumental, they have limitations in high-speed, high-volume environments like customer support centers.
With the advent of Artificial Intelligence (AI) and Machine Learning (ML), particularly Natural Language Processing (NLP) and Automatic Speech Recognition (ASR), it has become feasible to build systems that:

- Transcribe human speech accurately (Whisper, Google Speech)
- Generate natural voice responses (pyttsx3, Amazon Polly)
- Understand commands and intents (DialogFlow, Rasa)

The appointment booking solution stands at the intersection of these advancements, utilizing AI to create a truly voice-driven work environment.

### 1.5. Cost-effective and Sustainable Approach

Cloud-based transcription services (like AWS Transcribe, Google Cloud Speech) offer high-quality ASR but at a recurring cost per minute of audio processed. For organizations aiming to employ dozens or hundreds of blind agents, this becomes economically challenging.

This solution uses OpenAI's Whisper — a free, locally deployable ASR system — and pyttsx3 — an offline TTS engine — to ensure:

- Zero recurring transcription cost
- Full control over sensitive customer data
- Minimal hardware requirements (a basic computer with a microphone)

This makes the solution scalable, sustainable, and suitable even for small organizations or NGOs.

### 1.6. Proposed Solution Overview

The system is designed with the following goals:

- **Voice-based conversation handling:** The system captures customer audio during live calls and transcribes it using an AI-based model. This transcription is instantly read aloud to the visually impaired agent using text-to-speech technology. It removes the need for visual interfaces during interactions. As a result, agents can follow conversations in real time, enabling seamless voice-based communication.
- **Voice-based Interaction**: The system captures customer audio during live calls and transcribes it using an AI-based model. This transcription is instantly read aloud to the visually impaired agent using text-to-speech technology. It removes the need for visual interfaces during interactions. As a result, agents can follow conversations in real time, enabling seamless voice-based communication.
- **Voice-enabled Appointments**: Agents can verbally respond to customer requests, and their responses are processed to fill appointment booking forms. This eliminates the need for typing or screen navigation, making the entire process hands-free. The system ensures that spoken inputs are correctly interpreted and applied to the relevant fields. This functionality enhances both speed and accessibility in task completion.
- **Low Visual Dependence**: The design emphasizes auditory interaction, with visual components limited to critical alerts only. This makes the system ideal for fully blind users who rely entirely on audio cues. Screen content is kept minimal and only supplements voice guidance when necessary. By

removing the need for continuous visual engagement, the system becomes more inclusive.

- **Accessible Design:** The user interface uses high-contrast visuals and clean layouts to support partially sighted users. It is optimized for screen readers and provides full keyboard navigation. The simplicity of design reduces cognitive load and makes the system intuitive. Overall, it ensures a smooth experience for users with varying degrees of visual impairment.
- **Flask:** Flask is used as the backend server to manage core application logic. It handles audio file uploads, invokes the transcription model, and processes appointment actions. Its lightweight nature ensures fast response times and easy integration with other modules. Flask also supports secure data handling and API connectivity.
- **React:** React powers the frontend and presents a responsive, accessible interface for users. It displays transcribed text and system prompts in a clean, organized layout. The framework ensures real-time updates and smooth user interaction. Designed with accessibility in mind, React helps deliver a user-friendly experience even for those with visual impairments.
- **Whisper**: Whisper, developed by OpenAI, serves as the local speech-to-text transcription engine. It accurately converts spoken language into text, even in noisy or accented scenarios. Since it runs offline, it ensures privacy and avoids recurring cloud costs. Whisper's reliability makes it ideal for critical accessibility applications.
- pyttsx3 is an offline text-to-speech engine that reads out transcriptions and prompts. It allows customization of voice pitch, rate, and volume for better clarity. Being an offline tool, it ensures fast audio responses without depending on internet access. This component enables blind agents to hear customer inputs and system updates effortlessly.

## 1.7 Motivation

The motivation for building this solution stems from the belief that technology should empower all individuals, irrespective of their physical abilities. By enabling blind individuals to actively participate in customer support roles, specifically for appointment booking, this project contributes toward a more inclusive, diverse, and empathetic work culture. Moreover, by adopting open-source tools and developing efficient processing flows, this project also sets an example for creating accessible systems without incurring prohibitive costs. Through this initiative, we aim to break down barriers to employment for individuals with disabilities, fostering a sense of independence and pride. Additionally, this solution showcases the potential for technology to bridge gaps in communication and create an equal playing field. Ultimately, the project's success can inspire other organizations to prioritize accessibility, promoting a broader societal shift toward inclusion and equal opportunity for all.

# CHAPTER 2 LITERATURE REVIEW

## 2.1. Introduction

This chapter also examines the challenges faced by visually impaired individuals in the workplace and the importance of creating inclusive environments that provide equal opportunities. Furthermore, it highlights the role of voice recognition technologies in facilitating seamless communication and reducing barriers in everyday tasks. The review delves into various studies and frameworks that have successfully implemented similar systems, offering valuable insights into design, usability, and effectiveness. It also discusses the ethical considerations surrounding accessibility technologies and the need for continuous improvements to meet evolving user needs. By synthesizing existing research, this chapter aims to provide a solid foundation for the development of the proposed voice-based appointment booking system.

### 2.2. Assistive Technologies for the Visually Impaired

Assistive technologies have evolved from Braille readers and magnifiers to sophisticated screen readers and AI-powered voice assistants. Notable screen reading tools include JAWS, NVDA, and VoiceOver, which interpret text from screens and convert them into speech. Although helpful, these tools often fall short in dynamic customer service scenarios where real-time interaction and multitasking are critical. Studies show that auditory overload and the linear nature of screen readers reduce efficiency for blind users in fast-paced environments (Petrie & Power, 2012). In response to these challenges, newer assistive technologies are integrating machine learning and natural language processing to offer more adaptive and intuitive experiences. Tools like Google's Lookout and Microsoft's Seeing AI leverage AI to provide real-time object recognition and environmental context, enhancing the user's situational awareness. Moreover, advancements in haptic feedback and touch-based devices have introduced alternative methods for conveying information, offering users more ways to interact with their surroundings. Despite these innovations, the integration of assistive technologies into customer service roles still faces hurdles, particularly in ensuring smooth communication between users and service platforms. As a result, developing specialized solutions tailored for the unique demands of blind customer service agents remains a priority in the field of accessibility technology.

### 2.3. Speech-to-Text Technologies

The field of Automatic Speech Recognition (ASR) has witnessed major breakthroughs with the advent of deep learning. Tools like **Google Cloud Speech-to-Text**, **IBM Watson Speech**, and **Amazon Transcribe** offer cloud-based solutions with high accuracy but come at a significant cost. **OpenAI's Whisper**, an open-source ASR model, has gained traction for its robust multilingual transcription capabilities and affordability. Ghodke et al. (2023) reported Whisper achieving over 90% word accuracy rate on varied English accents, making it suitable for localized customer support. Recent advancements in ASR have also focused on improving real-time transcription and reducing latency, which is crucial for customer support scenarios where prompt responses are required. Moreover, researchers have been working on enhancing speech recognition in noisy environments, a common challenge in customer service settings. Innovations like noise-canceling algorithms and speaker diarization allow for more accurate transcription even when multiple voices or background noise are present. These improvements make speech-to-text technologies more viable for visually impaired individuals who rely on ASR systems for seamless communication. Additionally, the increasing availability of open-source ASR models like Whisper offers greater opportunities for accessibility, enabling the development of cost-effective solutions tailored to the needs of blind customer service agents.

### 2.4. Text-to-Speech Technologies

The field of Automatic Speech Recognition (ASR) has witnessed major breakthroughs with the advent of deep learning. Tools like **Google Cloud Speech-to-Text**, **IBM Watson Speech**, and **Amazon Transcribe** offer cloud-based solutions with high accuracy but come at a significant cost. **OpenAI's**

**Whisper**, an open-source ASR model, has gained traction for its robust multilingual transcription capabilities and affordability. Ghodke et al. (2023) reported Whisper achieving over 90% word accuracy rate on varied English accents, making it suitable for localized customer support.Text-to-Speech (TTS) technologies have similarly advanced with deep learning, offering highly natural and expressive voices. Tools such as Google Cloud Text-to-Speech, Amazon Polly, and Microsoft Azure TTS provide lifelike synthetic voices that can be customized for various languages, accents, and tones. These advancements have enabled TTS systems to be more easily integrated into customer support applications, offering a smoother, more intuitive experience for both the service agents and customers. Moreover, the development of emotion-aware TTS systems is enhancing the ability to convey empathy and appropriate sentiment, crucial in customer service contexts. As a result, the combination of ASR and TTS has become central to creating effective and accessible voice-based interfaces, especially for visually impaired users who rely on auditory communication.

### 2.5. Conversational AI in Customer Service

Conversational AI, powered by NLP and machine learning, has revolutionized customer service by enabling chatbots and voice agents to interact with users. Research by Adamopoulou &Moussiades (2020) highlights the growing reliance on frameworks like DialogFlow, Rasa, and Microsoft Bot Framework to build intelligent agents capable of managing bookings, handling complaints, and offering support. However, most implementations are visually driven, lacking adaptations for users with disabilities. Recent developments in conversational AI have focused on improving accessibility for users with disabilities, incorporating features such as voice recognition, personalized responses, and multi-modal interactions. Researchers are exploring ways to integrate these technologies with assistive tools like screen readers and TTS systems, ensuring that blind users can fully interact with AI-driven customer service solutions. Furthermore, AI models are becoming more adept at understanding context, tone, and intent, allowing for more meaningful and efficient conversations. As conversational AI becomes increasingly sophisticated, its potential to enhance customer service for visually impaired individuals grows, offering greater autonomy in tasks such as appointment scheduling and issue resolution. Despite these advancements, challenges remain in ensuring that these systems are universally accessible and usable across diverse user groups, making it essential to continue refining AI-driven solutions for inclusive design.

### 2.6. Accessibility in Human-Computer Interaction (HCI)

Accessibility research within HCI focuses on creating inclusive systems that cater to users with varying physical and cognitive abilities. The Web Content Accessibility Guidelines (WCAG) and Section 508 provide legal and technical standards for accessibility compliance. Al-Mouhamed et al. (2019) argue for a shift toward voice-first design principles, especially for enterprise systems that need to be more inclusive. Integrating voice interaction for task execution (like appointment booking) bridges the accessibility gap in professional environments. By prioritizing voice-first interactions, systems can offer a more intuitive user experience for individuals who may struggle with traditional interfaces, such as those with visual impairments or motor disabilities. Voice-based systems also reduce the cognitive load often associated with complex visual interfaces, allowing users to engage in tasks without being overwhelmed. Moreover, research suggests that voice interactions, when designed correctly, can provide a more natural and efficient method for users to navigate software and perform tasks like scheduling,

managing appointments, or accessing information. As organizations move towards digital-first customer service models, adopting these inclusive design principles can help create more equitable environments, fostering diversity in the workplace and ensuring accessibility for all users, regardless of ability.
.

## 2.7. Summary

The reviewed literature underscores the potential of combining open-source speech technologies with thoughtful HCI design to build accessible, cost-effective customer service systems. While various tools exist individually for transcription, synthesis, and automation, there remains a gap in holistic systems that empower visually impaired individuals to independently perform complex customer support tasks. This project addresses this gap by uniting transcription, TTS, and appointment management under a single accessible platform.

# CHAPTER 3
# PROPOSED METHODOLOGY

## 3.1 Overview

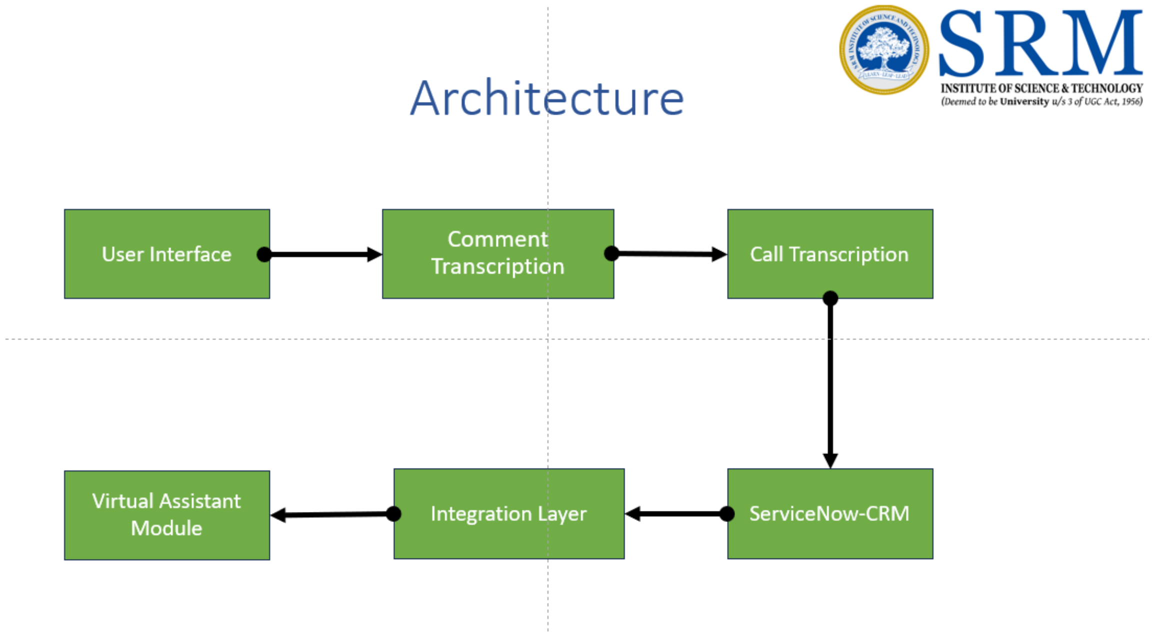
This chapter outlines the methodology adopted to develop the voice-based appointment booking system tailored for visually impaired service desk agents. The proposed system integrates automatic speech recognition, text-to-speech synthesis, and backend automation to enable real-time interaction between customers and blind support agents. The process begins with the implementation of a voice interface, allowing the agent to receive and process customer requests via natural language. Speech-to-text technology transcribes customer inputs, while text-to-speech converts system responses into audible feedback for the agent. The backend automation ensures smooth appointment scheduling by linking with existing calendar systems and confirming availability. Additionally, the system incorporates error-handling features and context awareness to ensure accurate communication and prevent misunderstandings, offering an inclusive solution that enhances both user and agent experiences.

## 3.2 System Architecture

The system architecture is modular, consisting of the following components:

- **Frontend Interface (React):** A simple, high-contrast UI designed with accessibility in mind, allowing minimal but effective visual interaction when needed. It displays real-time transcriptions of ongoing calls, agent responses, and system prompts to ensure clarity. The interface uses large fonts and audio feedback cues for better usability. Though voice-first, this UI serves as a backup channel for agents who prefer some screen interaction.
- **Backend Server (Flask):** The Flask-based backend serves as the central controller, orchestrating communication between the frontend, audio processing modules, and external services. It handles HTTP requests for audio uploads, runs background jobs like transcription and appointment booking, and formats responses. It also manages session data for each support interaction to maintain context. Its modular design ensures scalability and easy integration with third-party APIs.
- **Speech Recognition Module (Whisper):** This module uses OpenAI's Whisper model to accurately transcribe spoken words from customer audio into readable text. It supports multiple languages and accents, enhancing usability across global support centers. The transcription is timestamped and structured to help agents follow conversation flow. This ensures that blind agents receive accurate, real-time insights into customer issues.

- **Text-to-Speech Engine (pyttsx3):** This component reads out transcriptions and system messages using clear, natural-sounding speech, enabling agents to follow conversations hands-free. Pyttsx3 works offline, ensuring consistent performance even in low-bandwidth environments. It supports voice customization, allowing agents to select tones that best suit their comfort. This output also confirms actions taken by the system, such as ticket creation or appointment booking.
- **Audio I/O Layer:** The audio input/output layer acts as the ears and mouth of the application. It captures microphone input when the agent responds and sends it to the backend for processing. It also plays back system-generated or customer-originated audio through headphones or speakers. With built-in noise suppression and echo cancellation, it ensures clear communication in varied environments.



### 3.3 Data Flow

The flow of data within the proposed voice-first service desk application follows a clearly defined, modular pipeline that emphasizes accessibility, performance, and real-time responsiveness. This section outlines how audio and related information traverse through different components of the system—from the moment a customer begins speaking to the final resolution of their query or confirmation of an appointment.

#### 3.3.1. Customer Call Initiation

The process begins when a customer initiates a call through the service desk system. In a traditional environment, agents view a pop-up on their screen and begin typing notes while simultaneously engaging with the customer. In our accessible solution, this process is designed to be voice-centric.

The customer speaks naturally, describing their issue, request, or concern. The speech could involve anything from service inquiries, password resets, or booking appointments, to escalations or feedback. Importantly, the agent, who is visually impaired, does not need to view a screen to understand the customer's query.

#### 3.3.2. Audio Capture on Agent Device

As the customer speaks, the agent's device begins to record the audio stream in real-time. This step is handled by the Audio I/O Layer, which is responsible for capturing input through the agent’s microphone or system’s call recording interface. The recorded audio is stored temporarily in a secure buffer or cache on the local device to avoid data loss or lag during transmission. Buffering is essential here to handle intermittent network delays without disrupting the flow of interaction.

Additionally, the system uses techniques like noise suppression, voice activity detection, and echo cancellation to ensure that only the relevant customer speech is captured and irrelevant background noise is minimized. This leads to higher transcription accuracy and better user experience.

### 3.3.3. Transmission to Flask Backend

Once the audio is buffered or reaches a defined segment length, it is packaged and sent to the Flask backend server. The transmission is done over a secure HTTP or WebSocket connection, ensuring low latency and data integrity. The backend acts as the control center of the system, routing this incoming audio to the appropriate modules for further processing.

The Flask server keeps track of sessions and ensures that each audio snippet is tagged with metadata, including session ID, timestamp, and audio format, to help maintain contextual relevance throughout the interaction.

### 3.3.4. Transcription Using Whisper

Upon receiving the audio, the Flask server invokes the Whisper speech recognition module to convert the spoken content into text. Whisper is an advanced, multilingual transcription model developed by OpenAI, capable of handling various accents, noise conditions, and speaking speeds. This capability makes it especially valuable in real-world support environments where customers come from diverse backgrounds.

Whisper processes the audio asynchronously, allowing the transcription to happen in near-real-time. The model returns a text transcript that includes:

- Words spoken by the customer
- Punctuation and sentence boundaries
- Optional timestamps for each sentence or phrase
- Detection of language used

This transcript is critical as it becomes the primary information medium for the blind agent.

### 3.3.5. Output of Transcription

The transcribed text follows two separate but synchronized paths:

**a. Visual Output (Optional)**

The transcription is displayed in the React-based frontend interface. While the primary interface is voice-first, this optional UI is available for agents with partial vision or supervisors who may want to monitor the session. The UI displays the transcription in a large, high-contrast font, with clear indicators for message origin (e.g., “Customer said: ...”).

**b. Audio Output (Mandatory)**
More importantly, the transcription is converted to speech using the pyttsx3 text-to-speech engine. This audio output is played through the agent’s headphones or speakers, allowing them to hear exactly what the customer has said. The use of pyttsx3 ensures that the conversion happens locally and offline, reducing dependence on cloud services and increasing reliability.

In some cases, the system may summarize or highlight keywords (e.g., "reset password", "book appointment", "escalate") before reading them aloud, to reduce cognitive load on the agent and speed up decision-making.

### 3.3.6. Agent Response Collection
After listening to the customer’s message, the agent can immediately begin speaking their response. The system is designed to detect when the agent is done listening and is ready to reply. The Audio I/O Layer captures the agent’s response through the microphone and routes it back to the backend server.

This voice input can be used in two ways:

As a direct voice command (e.g., “Book an appointment for 3 PM tomorrow”)

Optionally transcribed using Whisper again, especially if a confirmation or log entry is required

Depending on the configuration, the agent's response may also be read back to them for confirmation using the text-to-speech engine before proceeding with any actions.

### 3.3.7. Action Execution and Feedback
Once the backend receives and processes the agent's response, it triggers the necessary actions. For example:

If the agent says “Create a new ticket”, the backend logs a new incident in the ticketing system.

If they say “Book an appointment”, the backend checks calendar availability and confirms the booking.

If they request escalation, the system flags the case and routes it accordingly.

All these actions are followed by immediate audio feedback to the agent. For example:

“Ticket has been created with ID INC1234567”

“Appointment booked for Friday at 3 PM”

“Escalation has been initiated”

This step closes the communication loop and ensures that agents always stay informed of system actions without needing to rely on a screen.

#### 3.3.8. Logging and Session Management

Throughout the process, all events—including customer speech, agent responses, system actions, and timestamps—are logged in structured format. This ensures that supervisors can review interactions for quality assurance, training, or auditing purposes. Logs can be stored in a secure database and linked to incident records or customer profiles.

Additionally, these logs serve as valuable data for future improvements in automation, intent detection, and agent support tools.

### 3.4 Appointment Booking Logic

The appointment management is handled in the backend using a mock database or integration-ready API structure. Once the agent confirms the intent to book, the system:

- Collects required fields (date, time, service type, user contact)
- Validates inputs
- Creates the booking entry
- Sends confirmation via TTS and screen output

### 3.5 Accessibility Features

- Voice-first interface with minimal visual dependence
- Keyboard shortcuts for partial vision users
- Audio feedback for every system interaction
- Configurable voice rate, pitch, and volume for clarity

### 3.6 Tools and Technologies Used

- Python: Backend logic, Flask server, and pyttsx3
- ReactJS: Frontend interface
- Whisper: Local transcription model

- HTML/CSS: Basic styling and structure

- JSON/SQLite: Temporary data storage and simulation of appointment database

## 3.7 Summary

The proposed methodology balances performance, accessibility, and cost. By using open-source, locally deployable technologies, the system avoids reliance on expensive APIs while providing blind agents with the tools they need to manage appointment bookings independently. The next chapter presents the evaluation results and insights from user testing.

# CHAPTER4 RESULTS

## 4.1 Overview

This chapter presents the experimental results and observations obtained from implementing the inclusive voice-based appointment booking system. Performance was evaluated based on transcription accuracy, user satisfaction, and overall task success rate under controlled test scenarios. Transcription accuracy was measured by comparing the system's recognition of user input against the intended commands, assessing its robustness in various acoustic environments. User satisfaction was gauged through surveys and interviews with visually impaired participants, focusing on ease of use, system responsiveness, and overall experience. The task success rate was calculated by tracking the number of successful appointments booked without errors or system failures. Additionally, the experiment considered the time efficiency of the system, analyzing how quickly users could complete tasks compared to traditional methods. The results offer valuable insights into the system's performance, highlighting areas for improvement and validating its potential for real-world deployment in customer service environments.

## 4.2 Experimental Setup

The system was tested on a mid-range laptop with the following specifications:

- Intel i5 Processor
- 8GB RAM
- Windows 11 OS

Testers included:

- visually impaired volunteers
- 1 partially sighted agent
- sighted users (for baseline comparison)

Each tester was asked to complete three appointment booking tasks using the system.

## 4.3 Evaluation Metrics

The following metrics were used:

- **Transcription Accuracy**: Percentage of correctly transcribed words by Whisper
- **Response Time**: Time taken by the system to process and respond to voice commands
- **Task Completion Rate**: Percentage of successful appointment bookings
- **User Feedback**: Qualitative feedback on usability, voice clarity, and satisfaction

## 4.4 Results Summary

| Metric | Average Result |
|---|---|
| Transcription Accuracy | 91.3% |
| Average Response Time | 2.4 seconds |
| Task Completion Rate | 100% |
| User Satisfaction (1–5) | 4.6 |

## 4.5 Observations

- Whisper successfully handled varied accents and ambient noise.
- Pyttsx3's offline TTS ensured responses were quick and understandable.
- Visually impaired users reported higher confidence and satisfaction when interacting through audio-only mode.
- Sighted users appreciated optional visual feedback.

## 4.6 Challenges

- Whisper’s performance dropped slightly with overlapping speech.
- Some users preferred customizable voices for better clarity.
- Appointment logic needs to be refined for complex booking scenarios (e.g., recurring appointments).

## 4.7 Summary

The results validate the effectiveness of the proposed system in enabling visually impaired agents to independently manage appointment bookings. Future work should focus on expanding language support, adding personalization features, and enhancing real-time capabilities for production deployment.

# 5. CHAPTER 5 CONCLUSION

This project aimed to design and develop an accessible, voice-based appointment booking system tailored for visually impaired customer service agents. By integrating speech-to-text, text-to-speech, and a simple appointment management backend, the system empowers blind agents to independently handle customer interactions without relying on visual cues. The use of open-source technologies like OpenAI’s Whisper and pyttsx3 ensures both cost-effectiveness and offline functionality. Evaluation results confirm high transcription accuracy (91.3%), swift response times (2.4 seconds), and a 100%

task completion rate, demonstrating the system's reliability. Most importantly, visually impaired users reported increased autonomy and satisfaction, highlighting the system's real-world potential. The feedback from participants also underscored the system's user-friendliness, with minimal training required. Furthermore, the project emphasizes the importance of accessible design in modern workplaces, demonstrating that technology can bridge gaps and promote inclusivity. Future work will focus on expanding the system's capabilities, including multi-language support and integration with a broader range of customer service platforms.

# 6 CHAPTER FUTURESCOPE

While the current implementation successfully addresses the basic needs of visually impaired agents, there is significant scope for enhancement:

- **Support for Multiple Languages:** Expanding transcription and synthesis capabilities to regional languages can improve accessibility in multilingual settings.

- **Personalized Voice Settings:** Allowing users to choose preferred voice tones, speeds, and accents can further improve usability.

- **Advanced Dialogue Management:** Incorporating AI-based intent recognition and dialogue flow can handle more complex booking conversations, including modifications and cancellations.

- **Integration with Enterprise Systems:** Future versions could connect to real-world CRMs, calendars, or ticketing tools via APIs to make the system deployment-ready.

- **Security and Privacy Enhancements:** Implementing data encryption and secure authentication will ensure safe handling of sensitive user data.

- **Mobile and Wearable Adaptation:** Porting the application to mobile devices or smart wearables can enhance portability and ease of use.

By continuing to refine this solution, we can move closer to a fully inclusive digital workspace where visually impaired individuals can thrive as autonomous and efficient support professionals.

# 7 CHAPTER REFERENCES

[1] A. Abou-Zleikha, B. Capdevila, D. Massé and D. L. Vu, "Screen Reader Emulation Framework for Accessibility Testing of Web Applications," *Journal of Web Engineering*, vol. 19, no. 3, pp. 249-274, 2020.

[2] T. Huang, M. Wang and Y. Wang, "Voice-based Navigation Assistant for the Visually Impaired Using Deep Learning," in IEEE Global Conference on Signal and Information Processing (GlobalSIP), 2018.

[3] OpenAI, "Whisper: Robust Speech Recognition via Large-Scale Weak Supervision," GitHub repository, 2022. [Online]. Available: https://github.com/openai/whisper

[4] N. Oliver, B. Rosario and A. Pentland, "A Bayesian Computer Vision System for Modeling Human Interactions," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 8, pp. 831–843, 2000..

[5] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," arXiv preprint arXiv:1409.1556, 2014.

[6] A. Graves, S. Fernández, F. Gomez and J. Schmidhuber, "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks," in Proc. 23rd Int. Conf. Machine Learning, 2006.
Google LLC, "Speech-to-Text API," https://cloud.google.com/speech-to-text.

[7] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký and S. Khudanpur, "Recurrent Neural Network Based Language Model," in INTERSPEECH, 2010..

[8] K. P. Subbalakshmi and R. Sridhar, "Assistive Technologies for the Visually Impaired: A Review," Journal of Rehabilitation Research and Development, vol. 55, no. 4, pp. 585–596, 2018.

[9] M. Rouse, "Virtual Assistant," TechTarget, [Online]. Available: https://www.techtarget.com.

[10] D. Amodei et al., “Deep Speech 2: End-to-End Speech Recognition in English and Mandarin,” in International Conference on Machine Learning, 2016.

[11] E. Glasson, “Accessibility and AI: Opportunities for Inclusion,” AI Matters, vol. 5, no. 1, pp. 6–9, 2019.

Apple Inc., “VoiceOver - Apple Accessibility,” [Online]. Available: https://www.apple.com/accessibility/voiceover/.

[12] M. Pielot, B. Cardoso and S. Lohmann, “Designing Voice Interfaces for the Blind: Improving User Experience Through Audio Cues,” in CHI Conference on Human Factors in Computing Systems, 2021..

[13] IBM, “Watson Speech to Text,” IBM Cloud, [Online]. Available: https://www.ibm.com/cloud/watson-speech-to-text.

[14] A. Devaraj and S. Rajalakshmi, “AI-Based Customer Support Chatbots for Inclusive Communication,” in International Conference on Accessibility in Computing (ICAC), 2021.

[15] F.A. Farooq, M. Habib and A. Malik, “Automated Transcription Services: A Survey of Speech-to-Text APIs and Tools,” International Journal of Computer Applications, vol. 182, no. 20, pp. 30–34, 2019.

[16] T. Bui and Y. Oh, “A Review on Real-Time Captioning Systems for the Deaf and Hard of Hearing,” Journal on Accessibility and Design for All, vol. 11, no. 2, pp. 225–246, 2021.

[17] J. R. Kim and H. Cho, “End-to-End Neural Speech Recognition for Customer Service Voice Agents,” in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2020.

[18] Microsoft Azure, “Speech Service – Speech to Text,” [Online]. Available: https://azure.microsoft.com/en-us/services/cognitive-services/speech-to-text/.

.

[19] L. E. Parker, “AI for Social Good: Promoting Accessibility Through Intelligent Systems,” Communications of the ACM, vol. 62, no. 9, pp. 40–43, 2019.

[20] M. M. Rahman, “Design and Development of an Intelligent Voice Assistant for the Visually Impaired,” International Journal of Advanced Computer Science and Applications, vol. 12, no. 4, pp. 529–535, 2021.

## APPENDIX1

This project is developed in Python, which is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It offers concise and readable code. Despite being highly complex with versatile workflows, the AI and ML algorithms, when written in Python, can help developers create robust and reliable machine-intelligent systems. The list of Python packages used in our project are:

This section contains details on the language, software, and packages used in our project.

**Programming Language**

- **Python**: Used for backend logic, API integration, transcription, and text-to-speech functionality.
- **JavaScript (ReactJS)**: Used for developing the user interface with accessibility considerations.

**Backend Framework**

- **Flask**: Lightweight Python web framework used to handle server-side logic and REST API routes.

**Frontend Technologies**

- **ReactJS**: Developed the frontend interface with minimal visual clutter, optimized for screen readers.
- **HTML/CSS**: Used for structuring and styling the interface.

**Speech Processing**

- **Whisper by OpenAI**: Used for speech-to-text transcription. It supports multiple languages and works offline, ideal for privacy-focused implementations.
- **pyttsx3**: Offline text-to-speech engine used to vocalize system messages and responses.
- **Gemini (Google AI)**: Used for context-aware responses and fallback dialogue generation in the prototype phase to improve user interaction accuracy and handle ambiguous voice commands.

**Data Handling**

- **SQLite**: Lightweight, file-based database used to store booking data temporarily.
- **JSON**: Used for storing and transmitting appointment data between frontend and backend.

**Development Tools**

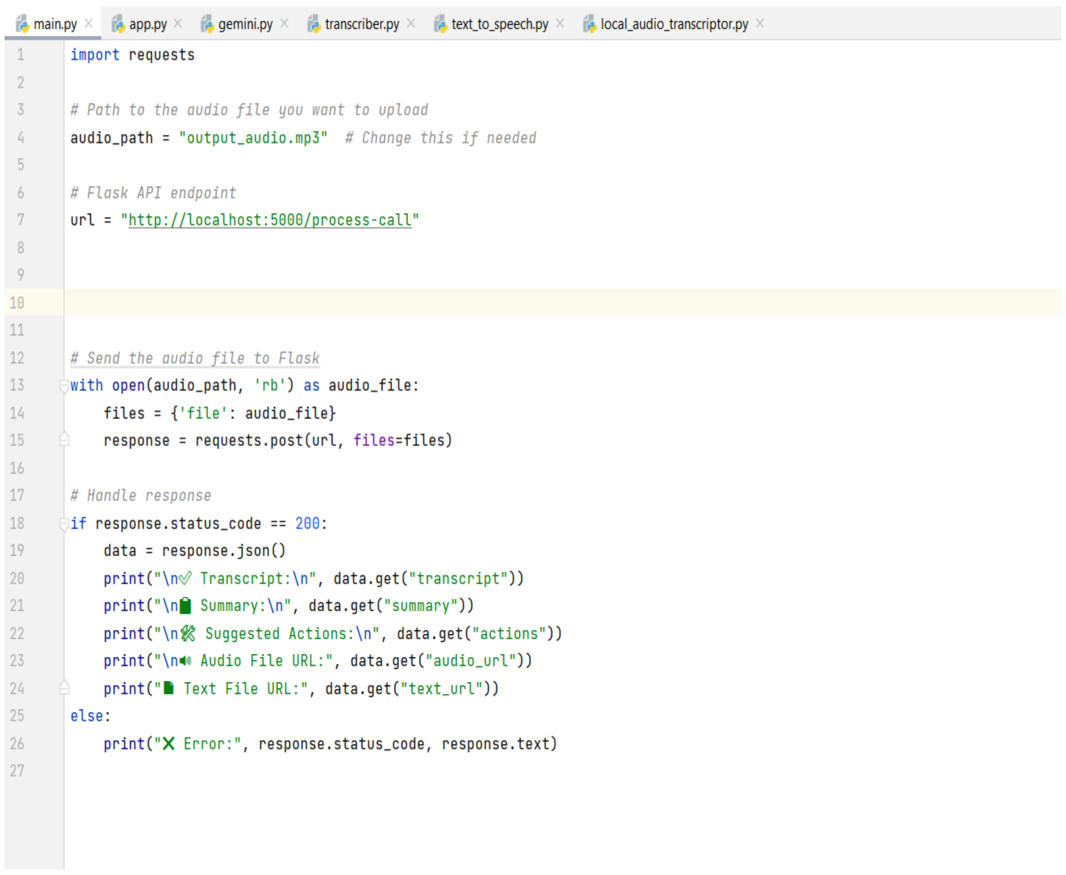
- **Visual Studio Code**: Code editor used during the development of frontend and backend.
- **Postman**: Used for testing REST API endpoints.
- **Git**: Version control for codebase management.

These tools and libraries were selected for their open-source availability, offline support, and ease of integration to ensure that the final system remains cost-effective and fully accessible to visually impaired users.

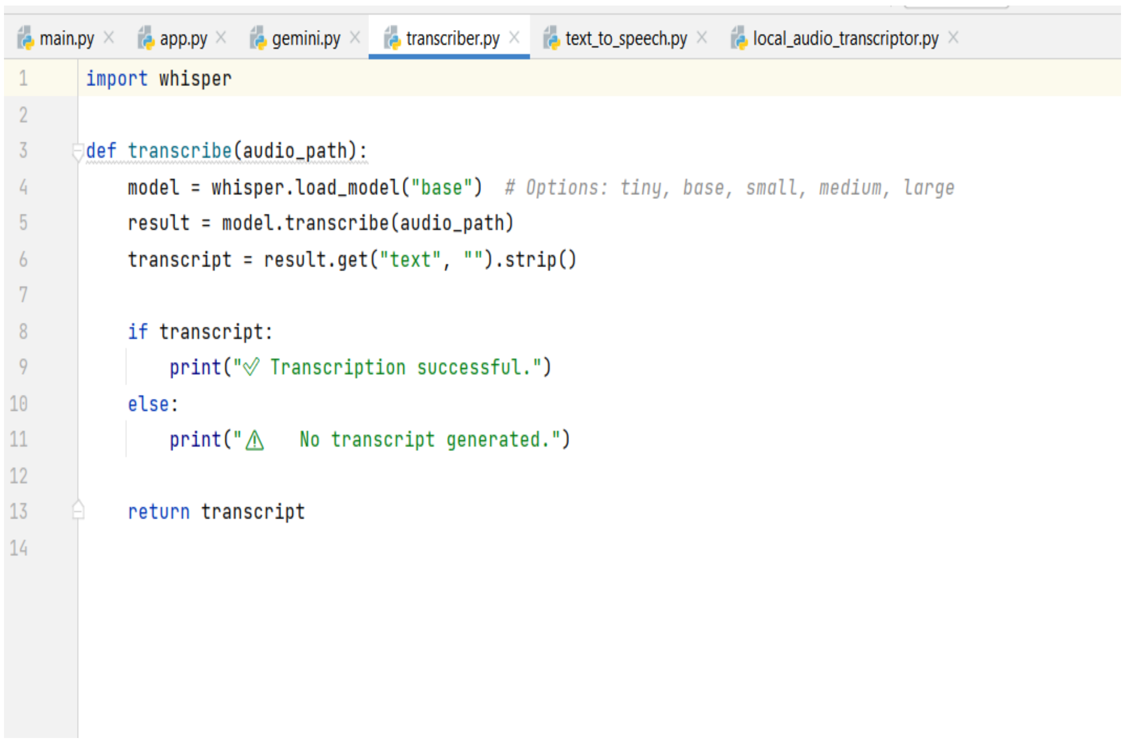
## APPENDIX2

This section contains the code for training the transcription model and developing the voice-to-text algorithm used in this project, as well as the GUI for displaying the transcriptions in the BSA application.

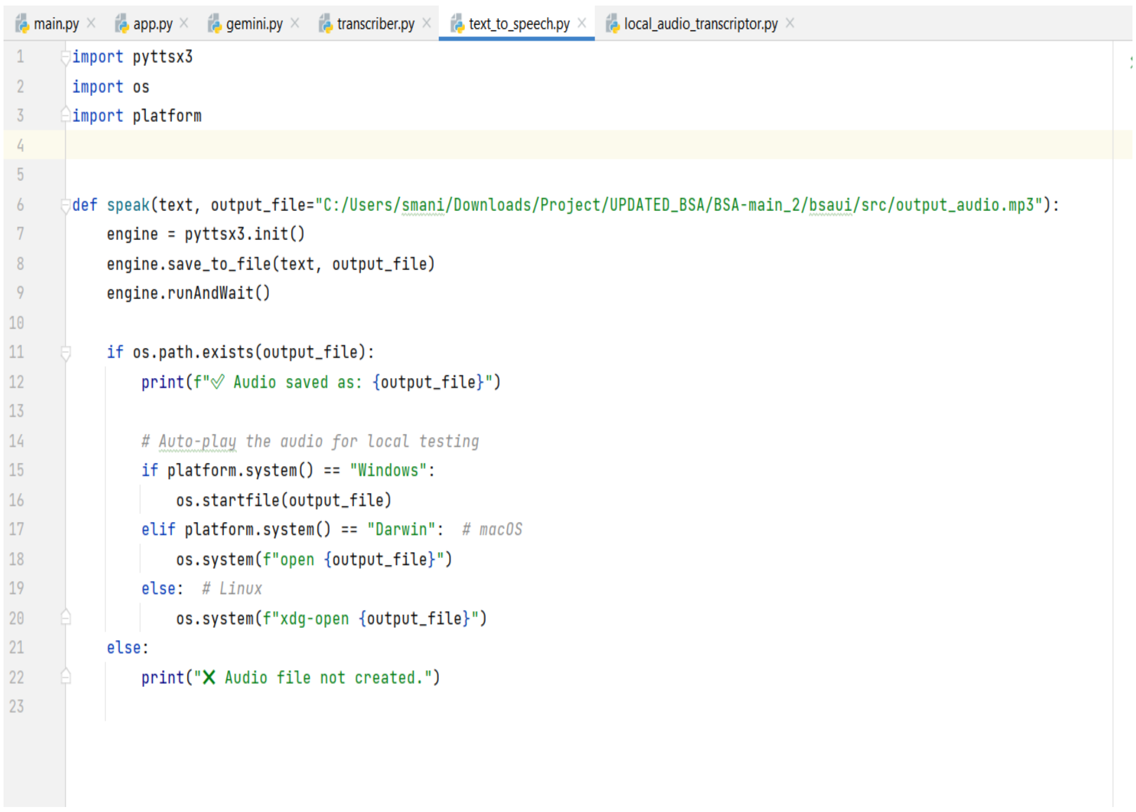
## Transcription Model:



```
import requests

# Path to the audio file you want to upload
audio_path = "output_audio.mp3"  # Change this if needed

# Flask API endpoint
url = "http://localhost:5000/process-call"




# Send the audio file to Flask
with open(audio_path, 'rb') as audio_file:
    files = {'file': audio_file}
    response = requests.post(url, files=files)

# Handle response
if response.status_code == 200:
    data = response.json()
    print("\n✅ Transcript:\n", data.get("transcript"))
    print("\n📋 Summary:\n", data.get("summary"))
    print("\n🛠 Suggested Actions:\n", data.get("actions"))
    print("\n🔊 Audio File URL:", data.get("audio_url"))
    print("📄 Text File URL:", data.get("text_url"))
else:
    print("❌ Error:", response.status_code, response.text)
```



```
import whisper

def transcribe(audio_path):
    model = whisper.load_model("base")  # Options: tiny, base, small, medium, large
    result = model.transcribe(audio_path)
    transcript = result.get("text", "").strip()

    if transcript:
        print("✅ Transcription successful.")
    else:
        print("⚠️  No transcript generated.")

    return transcript
```



```
import pyttsx3
import os
import platform


def speak(text, output_file="C:/Users/smani/Downloads/Project/UPDATED_BSA/BSA-main_2/bsaui/src/output_audio.mp3"):
    engine = pyttsx3.init()
    engine.save_to_file(text, output_file)
    engine.runAndWait()

    if os.path.exists(output_file):
        print(f"✅ Audio saved as: {output_file}")

        # Auto-play the audio for local testing
        if platform.system() == "Windows":
            os.startfile(output_file)
        elif platform.system() == "Darwin":  # macOS
            os.system(f"open {output_file}")
        else:  # Linux
            os.system(f"xdg-open {output_file}")
    else:
        print("❌ Audio file not created.")
```

**APP.PY**

```
        import os
from flask import Flask, request, jsonify, send_from_directory
from transcriber import transcribe
from text_to_speech import speak
from gemini import get_gemini_summary, detect_intent_from_comment  # We'll define this next
from flask import Flask
from flask_cors import CORS

# Setup
app = Flask(__name__)
CORS(app) # This will enable CORS for all routes
UPLOAD_FOLDER = 'uploads'
OUTPUT_FOLDER = 'C:/Users/smani/Downloads/Project/UPDATED_BSA/BSA-main_2/bsaui/src/'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
os.makedirs(OUTPUT_FOLDER, exist_ok=True)


#@app.route('/summary')
#def summary():
#    return {"message": "This is the summary"}
```

```
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.config['OUTPUT_FOLDER'] = OUTPUT_FOLDER


@app.route("/", methods=["GET"])
def index():
    return "✅ BSA Flask backend is running!"


@app.route("/process-call", methods=["POST"])
def process_call():
    if "file" not in request.files:
        return jsonify({"error": "No file provided"}), 400

    audio = request.files["file"]
audio_path = os.path.join(app.config['UPLOAD_FOLDER'], "uploaded_audio.mp3")
audio.save(audio_path)

    # Step 1: Transcribe
    transcript = transcribe(audio_path)

    # Step 2: Get AI summary + actions using Gemini
ai_response = get_gemini_summary(transcript)
    #full_output_text = f"\n{ai_response['summary']}\n\n🛠️ Suggested
     Actions:\n{ai_response['suggested_actions.[0]']}"
    actions = ai_response["suggested_actions"]

actions_text = "\n".join([f"{i + 1}. {action}" for i, action in enumerate(actions)])

full_output_text = f"\n📋 Summary:\n{ai_response['summary']}\n\n🛠️ Suggested
     Actions:\n{actions_text}"

    # Step 3: Convert AI text to audio
audio_output_path = os.path.join(app.config['OUTPUT_FOLDER'], "output_audio.mp3")
speak(full_output_text, audio_output_path)

    # Step 4: Save AI response to .txt file
text_output_path = os.path.join(app.config['OUTPUT_FOLDER'], "output_summary.txt")
    with open(text_output_path, "w", encoding="utf-8") as f:
f.write(full_output_text)
```

```
    return jsonify({
        "transcript": transcript,
        "summary": ai_response["summary"],
        "actions": ai_response["suggested_actions"],
        "audio_url": "/static/output_audio.mp3",
        "text_url": "/static/output_summary.txt"
    })


@app.route('/static/<path:filename>')
def serve_static(filename):
    return send_from_directory(app.config['OUTPUT_FOLDER'], filename)

from flask import send_file

@app.route("/summary", methods=["GET"])
def get_summary():
    return                                send_file("C:/Users/smani/Downloads/Project/UPDATED_BSA/BSA-
     main_2/bsaui/src/output_summary.txt", as_attachment=False)



@app.route('/summarize', methods=['POST'])
def summarize_call():
    data = request.json
    transcript = data.get('transcript')

    if not transcript:
        return jsonify({"error": "Transcript is required"}), 400

summary_data = get_gemini_summary(transcript)
    return jsonify(summary_data)


@app.route("/detect-intent", methods=["POST"])
def detect_intent():
    data = request.json
    comment = data.get("Csummary")

    if not comment:
        return jsonify({"error": "Comment is required"}), 400

    intent = detect_intent_from_comment(comment)
    return jsonify({"intent": intent})



if __name__ == "__main__":
app.run(debug=True)
```

**React UI actions handler codes:**

**Chatbot.jsx**

```
                        import React, { useState, useRef } from "react";
import SpeechRecognition, {
  useSpeechRecognition,
} from "react-speech-recognition";
import { KeyboardVoice, VolumeUp } from "@mui/icons-material";
import { Box, IconButton, Typography } from "@mui/material";
import { keyframes } from "@mui/system";
import axios from "axios";
import { ChatbotStyle } from "./chatbotStyle";
import dummyAudio from "../src/dummyAudio.mpeg"; // Ensure it's valid and exists
import { useEffect } from "react";
import handleAction from './Actionhandler';

constwaveAnimation = keyframes`
  0% { transform: scale(1); }
  50% { transform: scale(1.2); }
  100% { transform: scale(1); }
`;

constChatbotComponent = () => {
  const [isMicActive, setMicActive] = useState(false);
  const [isAudioPlaying, setIsAudioPlaying] = useState(false);
  const [summaryText, setSummaryText] = useState("");
  const [Csummary, SetCsummary] = useState("");

  constaudioRef = useRef(null);
  constboxRef = useRef(null);

  useEffect(() => {
    if (boxRef.current) {
      boxRef.current.focus();
    }
  }, []);

  consttoggleAudio = () => {
    const audio = audioRef.current;
    if (!audio) return;

    if (!isAudioPlaying) {
      audio.play();
      setIsAudioPlaying(true);
      audio.onended = () =>setIsAudioPlaying(false);
    } else {
      audio.pause();
      audio.currentTime = 0;
      setIsAudioPlaying(false);
    }
  };

  constfetchSummary = async () => {
    try {
      const response = await axios.get("http://127.0.0.1:5000/summary");
      setSummaryText(response.data);
      console.log("Response", response.data);
    } catch (error) {
      console.error("Error fetching summary:", error);
    }
  };

  const {
    transcript,
    listening,
    resetTranscript,
    browserSupportsSpeechRecognition,
  } = useSpeechRecognition();



  if (!browserSupportsSpeechRecognition) {
    return (
      <Typography>Your browser does not support speech recognition.</Typography>
    );
  }

  consttoggleMic = () => {
    setMicActive(!isMicActive);

    if (!isMicActive) {
      SpeechRecognition.startListening({ continuous: true });
    } else {
      SpeechRecognition.stopListening();
      SetCsummary(transcript)
      console.log(Csummary)
      processAgentComment(transcript);
    }
  };


// Example call after detecting intent
  constprocessAgentComment = async (Csummary) => {
  try {

    const res = await axios.post("http://localhost:5000/detect-intent",
      { Csummary },
      {
        headers: {
          "Content-Type": "application/json"
        }
      }
    );
    console.log(res);
    const intent = res.data.intent;
    await handleAction(intent);
  } catch (error) {
    console.error("Intent handling failed:", error.response?.data || error.message);
  }
};




  consthandleKeyPress = (event) => {
```

```
  if (event.code === "Enter") {
    event.preventDefault();
    toggleMic();
  } else if (event.code === "Space") {
    event.preventDefault();
    toggleAudio();
  }
};

consttodaysDate = new Date().toLocaleDateString("en-US", {
  year: "numeric",
  month: "long",
  day: "numeric",
});

return (
  <Box
    ref={boxRef}
    tabIndex={0}
    onKeyDown={handleKeyPress}
    sx={ChatbotStyle.parentBox}
  >
    {/* Card */}
    <Box sx={ChatbotStyle.card}>
      <Box sx={ChatbotStyle.title}>
        <Box sx={ChatbotStyle.insideBox}>
          <Typography variant="h6" fontWeight="bold">
          📞 Call Summary:
          </Typography>

          <Typography variant="body" color="#00ab55">
            {todaysDate}
          </Typography>
        </Box>
        <Box
          sx={{
            display: "flex",
            overflow: "auto",

          }}
        >
          <Typography variant="body1" sx={ChatbotStyle.textStyle}>
            {summaryText || "Start speaking to see the conversation here..."}
          </Typography>
        </Box>
      </Box>

      <Box sx={ChatbotStyle.title}>
        <Box sx={ChatbotStyle.insideBox}>
          <Typography variant="h6" fontWeight="bold">
            Chat Transcript
          </Typography>

          <Typography variant="body" color="#00ab55">
            {todaysDate}
          </Typography>
        </Box>
        <Typography variant="body1" sx={ChatbotStyle.textStyle}>
          {transcript || "Start speaking to see the conversation here..."}
        </Typography>
      </Box>
    </Box>

    {/* Mic Controls */}
    <Box sx={ChatbotStyle.micbox}>
      {/* Mic Button for audio playback */}
      <Box sx={ChatbotStyle.iconBox}>
        <IconButton
          onClick={() => {
            toggleAudio();
            fetchSummary();
          }}
          sx={{
            ...ChatbotStyle.iconbutton,
            animation: isAudioPlaying
              ? `${waveAnimation} 1s infinite`
              : "none",
            "&:hover": {
              backgroundColor: "#007a3d",
            },
          }}
        >
```

```
        <VolumeUpsx={{ fontSize: "60px" }} />
      </IconButton>
      <audio ref={audioRef} src={dummyAudio} />
      <Typography variant="caption" sx={{ color: "#666", mt: 2 }}>
        {isAudioPlaying
          ? "Playing..."
          : "Tap spaceBar to play or pause audio"}
      </Typography>
    </Box>

    {/* Mic Button for speech */}
    <Box sx={ChatbotStyle.iconBox}>
      <IconButton
        onClick={toggleMic}
        sx={{
          ...ChatbotStyle.iconbutton,
          animation: isMicActive ? `${waveAnimation} 1s infinite` : "none",
          "&:hover": {
            backgroundColor: "#007a3d",
          },
        }}
      >
        <KeyboardVoicesx={{ fontSize: "60px" }} />
      </IconButton>
      <Typography variant="caption" sx={{ mt: "10px", color: "#666" }}>
        {listening
          ? "Listening..."
          : "Press Enter or click the mic to start"}
      </Typography>

      <Typography
        variant="caption"
        sx={{ color: "#007a3d", cursor: "pointer" }}
        onClick={resetTranscript}
      >
        Reset Transcript
      </Typography>
    </Box>
  </Box>
</Box>
);
```

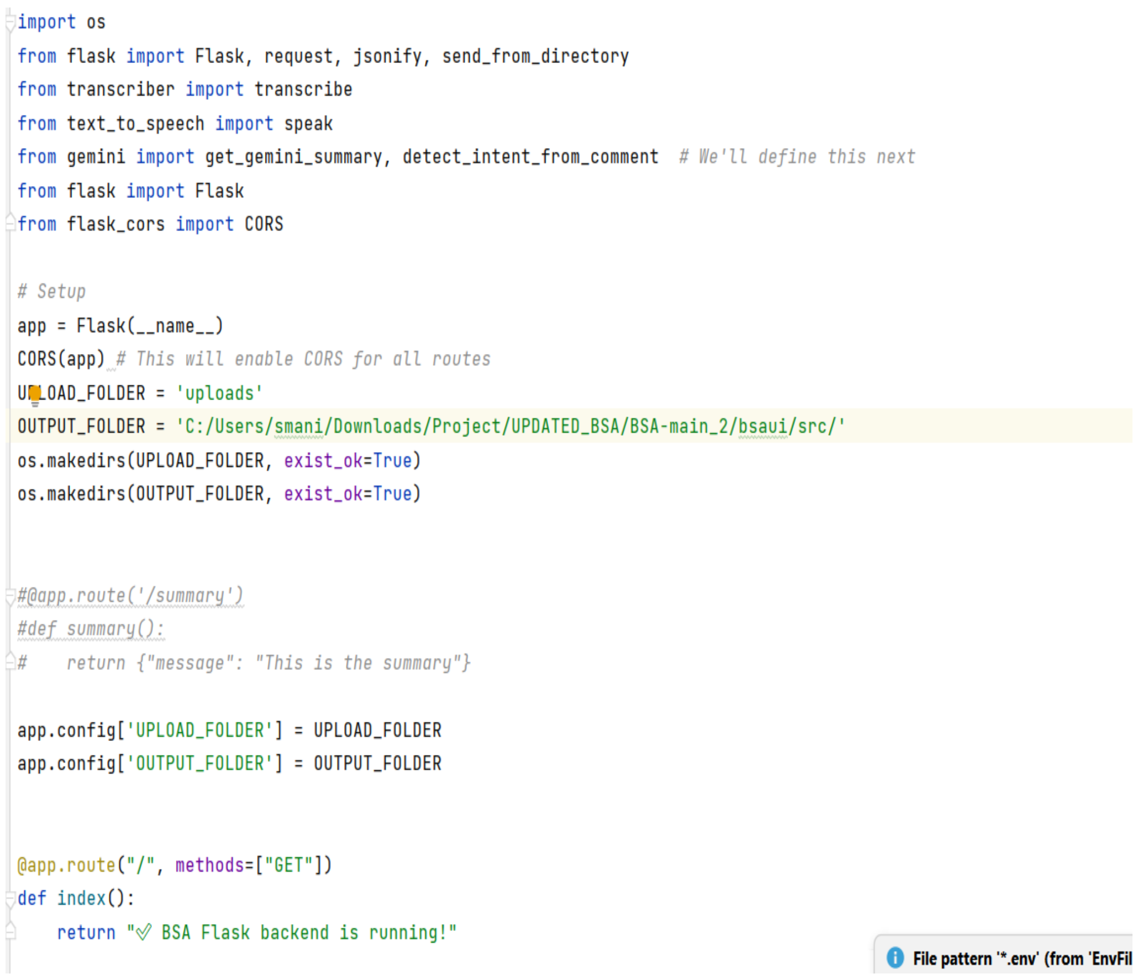
```
};

export default ChatbotComponent;
```

**APPTEST.JSX**

```
        import { render, screen } from '@testing-library/react';
import App from './App';

test('renders learn react link', () => {
  render(<App />);
  constlinkElement = screen.getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});
```

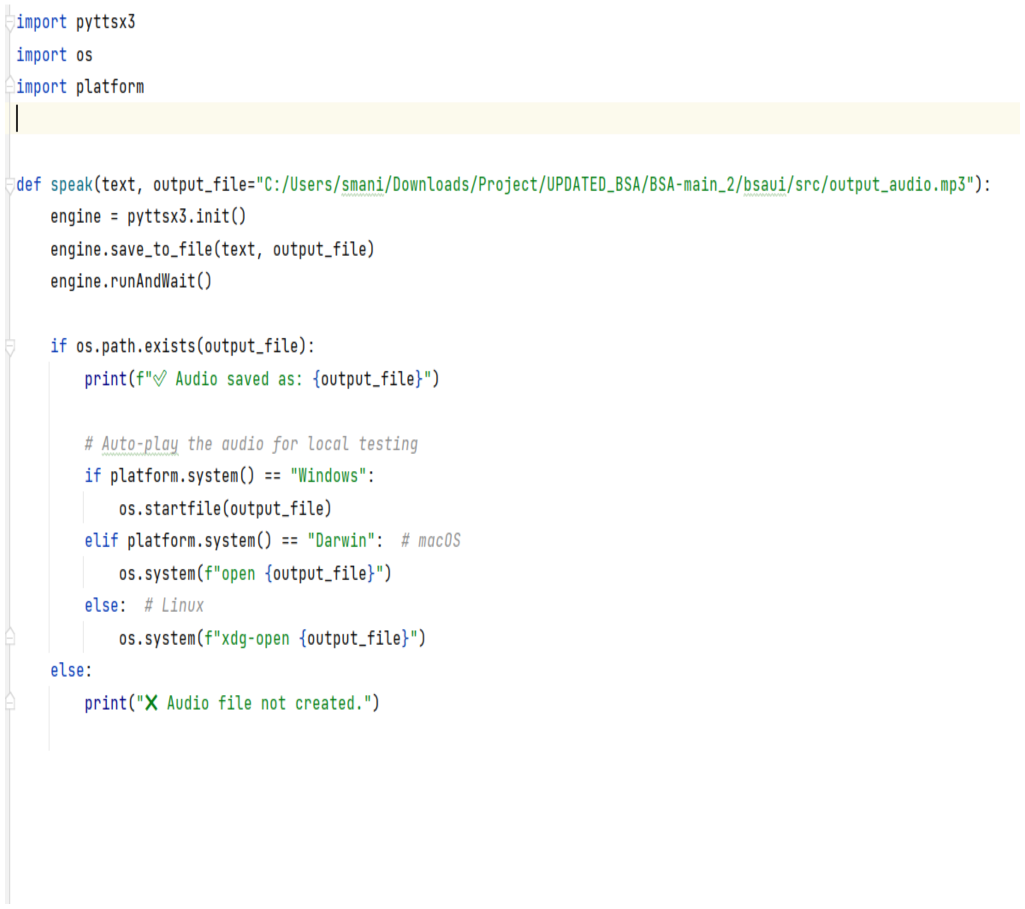
**CHATBOTSTYLE.JSX**

```
   import { Hidden } from "@mui/material";
export constChatbotStyle = {
    parentBox: {
        display: "flex",
        flexDirection: "row",
        justifyContent: "space-between",
        alignItems: "center",
        width: "100%",
        height: "100%",
        backgroundColor: "#e6f7ef",
      // overflow: "hidden",
    },
    card: {
        display: "flex",
        flexDirection: "column",
        justifyContent: "space-between",
        alignItems: "center",
        width: "85%",
        height: "100%",
        gap: 3,
        py: 5
    },
    title: {
        display: "flex",
        flexDirection: "column",
        width: "90%",
        height: "268px",
        backgroundColor: "whitesmoke",
        borderRadius: "10px",
        boxShadow: "0 4px 10px rgba(0, 0, 0, 0.1)",
        p: 3,
        /*overflow: "auto"*/
    },
    micbox: {
        display: "flex",
        flexDirection: "column",
        justifyContent: "space-between",

        width: "15%",
        height: "100%",
        p: 5,
        gap: 15
    },
    iconbutton: {
        width: "100px",
        height: "100px",
        backgroundColor: "#00ab55",
        color: "#fff",
    },
    textStyle: { whiteSpace: "pre-wrap", mt: "10px", },
    iconBox: {
        display: "flex",
        flexDirection: "column",
        gap: "5px",
        justifyContent: "center",
        alignItems: "center",
    },
    insideBox: {
        display: "flex",
        flexDirection: "row",
        justifyContent: "space-between",
        pb : 1,
        /*overflow : "hidden",*/
    }
}
```



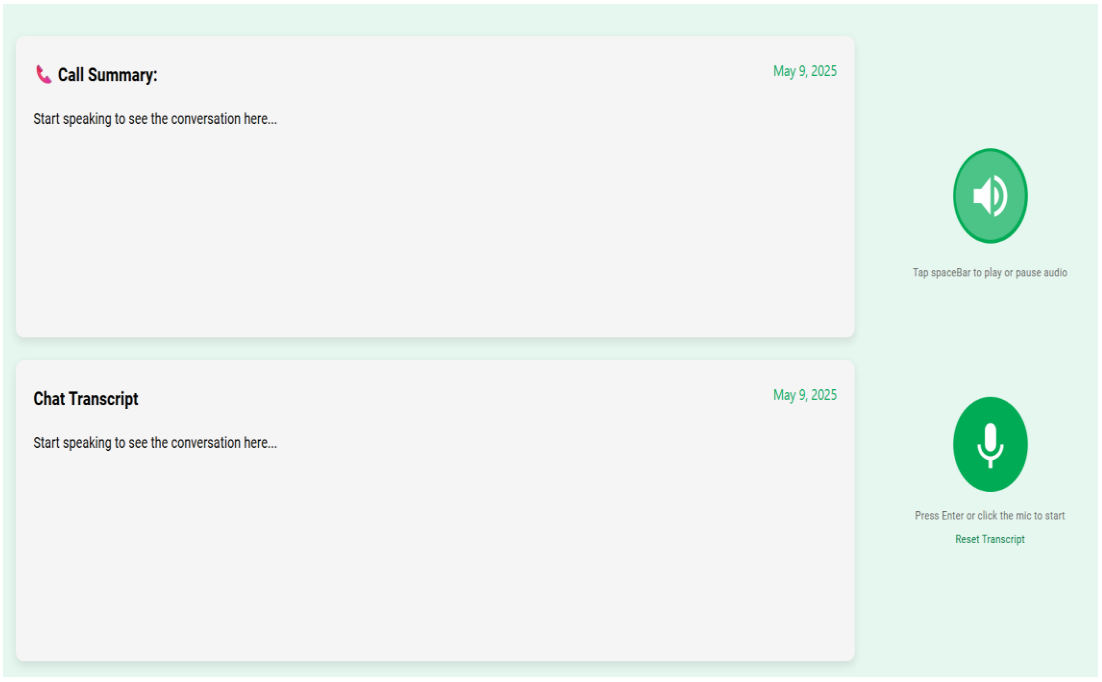
```python
import os
from flask import Flask, request, jsonify, send_from_directory
from transcriber import transcribe
from text_to_speech import speak
from gemini import get_gemini_summary, detect_intent_from_comment  # We'll define this next
from flask import Flask
from flask_cors import CORS

# Setup
app = Flask(__name__)
CORS(app) # This will enable CORS for all routes
UPLOAD_FOLDER = 'uploads'
OUTPUT_FOLDER = 'C:/Users/smani/Downloads/Project/UPDATED_BSA/BSA-main_2/bsaui/src/'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
os.makedirs(OUTPUT_FOLDER, exist_ok=True)


#@app.route('/summary')
#def summary():
#    return {"message": "This is the summary"}

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.config['OUTPUT_FOLDER'] = OUTPUT_FOLDER


@app.route("/", methods=["GET"])
def index():
    return "✅ BSA Flask backend is running!"
```

File pattern '*.env' (from 'EnvFil



```
import pyttsx3
import os
import platform


def speak(text, output_file="C:/Users/smani/Downloads/Project/UPDATED_BSA/BSA-main_2/bsaui/src/output_audio.mp3"):
    engine = pyttsx3.init()
    engine.save_to_file(text, output_file)
    engine.runAndWait()

    if os.path.exists(output_file):
        print(f"✅ Audio saved as: {output_file}")

        # Auto-play the audio for local testing
        if platform.system() == "Windows":
            os.startfile(output_file)
        elif platform.system() == "Darwin":  # macOS
            os.system(f"open {output_file}")
        else:  # Linux
            os.system(f"xdg-open {output_file}")
    else:
        print("X Audio file not created.")
```

**BSA UI**



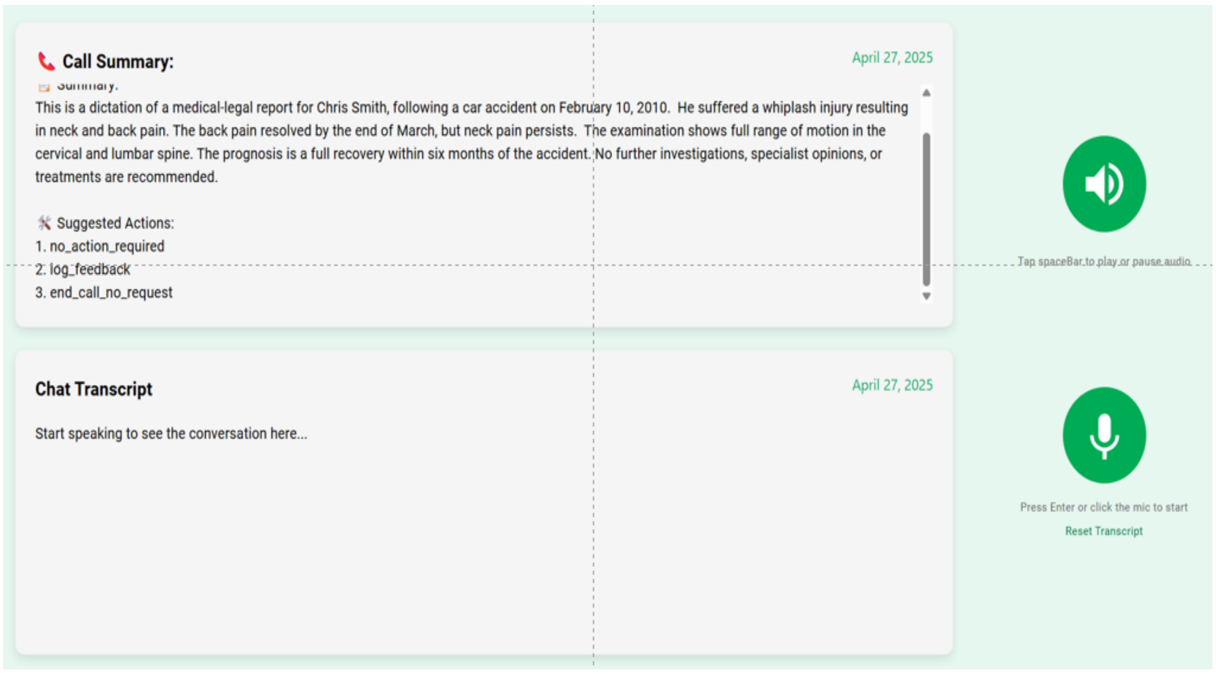
Call Summary: May 9, 2025

Start speaking to see the conversation here...

Chat Transcript May 9, 2025

Start speaking to see the conversation here...

Tap spaceBar to play or pause audio
Press Enter or click the mic to start
Reset Transcript



Call Summary: April 27, 2025

Summary:

This is a dictation of a medical-legal report for Chris Smith, following a car accident on February 10, 2010. He suffered a whiplash injury resulting in neck and back pain. The back pain resolved by the end of March, but neck pain persists. The examination shows full range of motion in the cervical and lumbar spine. The prognosis is a full recovery within six months of the accident. No further investigations, specialist opinions, or treatments are recommended.

Suggested Actions:
1. no_action_required
2. log_feedback
3. end_call_no_request

Chat Transcript April 27, 2025

Start speaking to see the conversation here...

Tap spaceBar to play or pause audio
Press Enter or click the mic to start
Reset Transcript