|  |
| --- |
| Complex Engineering Problem |
| SoundSweep |
| Noise Reduction Mobile Application |
|  |
|  |
|  |

|  |
| --- |
|  |

**Pp**

**1 |**

1. **Project Overview:**

**A Real-Time Personalized Noise Reduction Smartphone App for Hearing Enhancement.**

This project report introduces **SoundSweep,** a real-time, low-latency noise reduction app designed to enhance hearing on smartphone platforms.

Developed with the purpose of empowering users in noisy environments, **SoundSweep** allows individuals to record audio and apply advanced noise reduction techniques to filter out background sounds, providing clearer sound for improved focus on primary voices and sounds.

**SoundSweep** offers users the ability to easily capture and process audio recordings. Through an intuitive interface, users can select files for noise reduction and conveniently store the processed audio in the device. This functionality makes it a practical tool for individuals frequently in noisy spaces, such as open offices, public transport, noisy study lecturers or busy public areas.

1. **Identification of Real-World Problem:**

**Problem Overview:**

In our fast-paced **technological world,** clear sound is essential for effective communication and productivity. Unfortunately, background noise has become a pervasive issue, interfering with sound clarity and impacting personal interactions, professional environments, and accessibility needs. Individuals often struggle to communicate or understand others in noisy surroundings, a challenge that is exacerbated when technology transmits these unwanted sounds. This project aims to address the critical need for noise reduction technology, specifically targeting **smartphone applications that can filter out noise in real time.**

**Consequences of Poor Voice Clarity in Educational and Environmental Contexts:**

In **educational settings**, background noise disrupts students’ focus and comprehension, making it challenging to retain information and leading to ineffective communication from instructors who struggle to convey information clearly amidst distractions.

Poor audio also limits accessibility, especially for students in online learning environments or those with auditory processing challenges.

Similarly, in everyday environments, noise from **traffic, crowds, or construction** creates obstacles to effective communication and degrades the quality of recordings, making them difficult to use for study or professional purposes. Overall, poor voice clarity reduces focus and engagement in settings like libraries or study spaces, where clear audio is essential for concentration.

**Importance of Clear Sound in Educational and Environmental Applications:**

Clear audio is crucial in educational contexts, where it enhances students’ **understanding, participation**, and **retention** of material in both live and recorded lectures. Noise-free audio quality ensures that complex content is accessible and easier to study without extraneous distractions. In everyday settings, reducing background noise improves communication, making conversations more effective in public or noisy spaces. Additionally, in shared spaces such as libraries or parks, clear sound helps maintain focus and productivity, while in outdoor recreational contexts, it preserves essential sounds for a more enjoyable and immersive listening experience across various environments.

1. **Problem Solution:**

To address the pervasive issue **of background noise in audio recordings**, we have developed a **real-time mobile application** that captures audio, **applies noise reduction**, and allows users to listen to both the original and **noise-reduced versions**. This solution is ideal for various settings, including education, daily life, and professional environments where clear audio quality is essential.

**Our application uses two key filtering techniques to minimize unwanted sounds and maintain speech clarity:**

1. **Highpass Filter (f = 300 Hz):**

This filter removes frequencies **below 300 Hz**, targeting low-frequency sounds such as hums, rumbles, and other background noises **(e.g., fans or engines).**

By filtering out these frequencies, the application preserves essential audio

Elements, particularly speech, while reducing disruptive low-frequency

noise.

1. **Lowpass Filter (f = 3000 Hz):** This filter removes frequencies **above 3000 Hz,** reducing high-pitched noises such **as clicks, hisses, or sharp sounds**.

This further refines audio quality by eliminating high-frequency noise while

maintaining intelligibility.

By combining these filters, the app retains audio **frequencies between 300 Hz and 3000 Hz,** which is the range where human voices primarily reside. This frequency range is optimal for preserving clear speech while minimizing disruptive background noise.

To ensure the app's effectiveness**, we tested it using publicly available datasets** featuring various types of noise.

**This noise reduction app is versatile and can be used in diverse settings, such as:**

1. **Educational Environments**: In classrooms or lecture halls, users can record lectures on their phones and use the SoundSweep app to reduce background noise. Users simply provide the noisy recording, and the app saves a clear version for future listening.
2. **Noisy Environments:** Users can record audio or receive audio messages in noisy settings, such as during travel or in crowded places. The app reduces noise, providing a clear and understandable message.
3. **Everyday Use:** The app can be used to reduce ambient noise in daily recordings, making it suitable for note-taking, capturing meetings, and other practical applications.

**App Functionality:**

1. **Sound Input** Users can input noisy sound recordings from device storage into the SoundSweep app.
2. **Noise Reduction:** With a simple tap on the "Reduce Noise" button, the app processes the audio to filter out unwanted sounds.
3. **Playback Options:** Users can listen to both the original and noise-reduced versions for comparison.
4. **Save:** Users have the option to save the noise-reduced audio within the app. Additionally, users can access and listen to all saved audio files in the app.
5. **Responsive User Interfaces:**
6. **Data Storage Strategy:**  
   For data storage, we use an **SQLite database** to manage the names and paths of reduced audio files. SQLite stores **this information in string format**, allowing our app to efficiently retrieve the audio files when needed. When a user wants to listen to a reduced audio file, the app accesses the path stored in SQLite, searches the device storage, and plays the corresponding audio.

**We chose SQLite for several reasons:**

1. **Lightweight:** SQLite has a small footprint and uses minimal storage and memory, making it easy to integrate into our app without significant overhead.
2. **Serverless:** Being serverless means that SQLite operates without the need for a separate server, simplifying deployment and reducing complexity.
3. **Offline Use:** SQLite is embedded directly within the app, enabling data storage and retrieval even without an internet connection. This feature ensures that our application can function seamlessly across all platforms and in various environments.
4. **Efficient for Small Datasets:** Our app primarily stores basic information, such as file names and paths. SQLite's efficient handling of small datasets allows for quick read and writes operations, perfectly aligning with our app’s requirements.

In summary, SQLite provides a robust and efficient solution for managing audio file data in our application while ensuring optimal performance across all platforms.

1. **APIs, Packages, and Plug-ins Used:**
2. **FlutterFFmpeg:** We use **FlutterFFmpeg** for noise reduction. This powerful package allows us to apply various audio processing techniques, ensuring that the audio files maintain clarity and quality after noise reduction. Its flexibility and extensive functionality make it ideal for our needs.
3. **AudioPlayers:**  The AudioPlayers package enables seamless audio playback within our app. It provides a simple interface for playing, pausing, and managing audio files, making it essential for delivering a smooth user experience when listening to reduced audio files.
4. **File\_Picker:**  We use the File\_Picker package to allow users to easily select audio files from their device storage. This package streamlines the process of choosing audio files, enhancing user convenience and interaction with our application.
5. **Path Provider:** The Path Provider package helps us determine the correct directory for saving audio files. By providing access to commonly used directories on the device, it simplifies file management and ensures that our app can store audio files efficiently.
6. **Sqflite:**  Sqflite is utilized for the storage of audio file names and paths. This package allows us to manage SQLite databases in a straightforward manner, enabling efficient data retrieval and storage operations that are crucial for our app's functionality.
7. **Provider:**  We employ the Provider package for state management in our application. This package allows us to efficiently manage the state across different components of our app, ensuring a responsive and dynamic user interface as audio files are processed and played.
8. **Development Challenges: Issues and Bugs Resolved:**
9. **Algorithm Trials:**  **Initially, we tried two different noise reduction algorithms (ANLMND and FFTDN)** with various parameters but did not achieve satisfactory results. We eventually implemented a method from a YouTube tutorial focusing on high and low-frequency noise reduction, which proved to be more effective.
10. **Audio Overwriting**: We encountered an issue where audio files could not be reduced a second time because overwriting was not enabled, and temporary file names were hardcoded in the code. To resolve this, we included the -y command to allow overwriting and later saved the files with custom names.
11. **Play/Pause Button Functionality**: The play and pause buttons were not updating correctly according to the audio state due to the incorrect placement of the set State function. After restructuring the code, the buttons now reflect the correct audio state.
12. **Understanding Provider and SQLite Integration:**  It took considerable time to grasp how to effectively use the Provider package alongside SQLite. With assistance from GPT, we learned that database methods should be called within the provider methods, while the main code should only invoke the provider methods.
13. **Saved Audio List Icon Issue**: We faced a problem where clicking one play/pause icon changed all icons in the saved audio list. This was caused by a globally scoped Boolean variable. By scoping the variable locally, we resolved the issue.
14. **Delete Icon Functionality:**  The delete icon was not updating the list properly due to incorrect placement of the remove function. We adjusted the code to ensure that the UI reflects deletions accurately.
15. **Temporary Path Debugging:**  To facilitate debugging, we added a temporary text field to display the path where the audio files are saved, which helped us track file locations more effectively.
16. **File Handling Exceptions:**  We encountered numerous exceptions related to file handling. To manage this, we implemented try-catch blocks extensively to capture errors and utilized print statements to monitor the workflow and identify issues.
17. **Tutorial Availability:** Since FlutterFFmpeg is primarily focused on video and image processing, finding relevant tutorials for audio applications was challenging. This made it difficult to locate helpful resources, but GPT provided significant assistance in navigating these issues.