User and Product Manual Instructions

This document is a template of a user and product manual. The client may wish to make improvements on the prototype or need to fix it if something goes wrong or another group of students may work to make a more rugged prototype. The document needs to be clear for someone else who is not an engineer **to use, maintain or reproduce the project**. Include as many images and diagrams as possible for a better understanding. Keep it plain, simple, visual and logical.

In general, if you are not sure exactly what to include, imagine that this document was the only thing that you had. Imagine also that your job was to add a new feature or recreate the project that is described in your document. What would you need to know?

Only include details relating to your final prototype.

Template conventions:

- Remove all red text, it is only there to guide you
- Remove this page (instructions)
- Replace all instances of <xxx> with the appropriate information for your group, for example you could replace <System Name (Acronym)> by The Amazing Product (TAP)
- Save this document as 'User and Product Manual_group number' instead of Deliverable X so that others know what it represents when they see it in MakeRepo

GNG2101

Design Project User and Product Manual

SpeechSyncSolutions

Submitted by:

Hasnain Sahibzada, 300220482

Orel Benkarmona , 300160407

Arjun Jatania , 300242065

Avneesh Chaudhary 300120325

Ali Allouche, 300373757

Yingying, 300186973

10th March, 2024

University of Ottawa

Table of Contents

1 Introduction

This document outlines the design and implementation of a system aimed at enhancing the accessibility of communication devices for a client, particularly focusing on improving voice recognition capabilities on their mobile phone. The primary goal is to address the challenges faced by the client in using voice recognition for essential functions such as making calls, checking the time, setting timers, and potentially accessing other functionalities with minimal reliance on voice commands. The solution aims to be user-friendly, require minimal maintenance, and be accessible independently by the client.

The document is structured to cover the problem definition, concept development, detailed design, Bill of Materials, project planning, and a sustainability report. It begins with a detailed problem definition, highlighting the issues with the current speech-to-text design, including decreased accuracy in noisy environments and slow text generation with certain accents or dialects. The document then outlines the steps planned to improve the design, including updating the speech recognition engine, optimizing environmental noise processing, and enhancing the speed of text generation from speech.

The intended audience for this document includes the development team responsible for enhancing the voice recognition system, stakeholders involved in the project, and the client who will benefit from the improved accessibility. The document also addresses security, safety, and privacy considerations associated with the use of the User and Product Manual, emphasizing the importance of data protection and user privacy in the development and implementation of the system.

In summary, this document serves as a comprehensive guide for the development of a system that significantly improves the client's ability to use their mobile phone independently, focusing on enhancing voice recognition capabilities. It provides a structured approach to problem-solving, from identifying the issues to outlining the steps for improvement, ensuring that the final product meets the needs of the client and adheres to the highest standards of security, safety, and privacy

2 Overview

The problem we aim to address is the significant barrier to communication faced by individuals with speech impairments, particularly in the context of using voice recognition technology. This issue is crucial because it directly impacts their ability to participate fully in social, educational, and professional environments, where effective communication is essential. The fundamental needs of these users include a reliable, accurate, and accessible tool that can convert their spoken words into text in real-time, enabling them to communicate more easily and independently.

Our product stands out from others by leveraging advanced Whisper AI technology, which is specifically designed for real-time speech-to-text conversion. This technology ensures fast and accurate transcription, making our solution particularly suited for individuals with speech impairments who require immediate and precise text output from their speech. The key aspects that differentiate our product include its focus on real-time functionality, the use of Whisper AI for enhanced accuracy, and its operation in a local environment to minimize latency and enhance efficiency.

The key features of our product include:

- Real-time speech-to-text conversion.
- Advanced Whisper AI technology for fast and accurate transcription.
- Local operation to minimize latency.

In terms of architecture, our system is designed to be user-friendly and accessible. It operates locally on the user's device, ensuring that the transcription process is fast and efficient. The user accesses the system through a graphical user interface (GUI), which allows for easy interaction and control. There are no special conditions for using the system, making it accessible to a wide range of users with speech impairments.

Our solution aligns with the goal of promoting inclusivity and accessibility by providing a tool that empowers individuals with speech impairments to communicate more effectively. The feasibility of our solution is supported by the advanced capabilities of Whisper AI, which has been recognized for its performance in real-time speech-to-text applications.

3 Getting started

Before installing and running the Live Transcription System, ensure your computer meets the following requirements:

- Operating System: Windows, macOS, or Linux. • Python Version: 3.8 or later.
- Required Libraries: speech recognition, numpy, torch, and whisper.
- Hardware: A functional microphone is required for capturing audio.

To set up the Live Transcription System, start by installing the necessary Python packages. Open a terminal or command prompt and execute the following command:

```
pip install speechrecognition numpy torch whisper
```

This command installs all the dependencies required by the transcription system

4 Using the System

Running the Live Transcription System is straightforward. Use the following command to start the system with default settings:

```
python live_transcription.py --model small --english True
```

This command initiates the system using the ‘small‘ Whisper model, configured for English language transcription.

Options The system includes several command-line options for customization:

- model: Specifies the model size. Larger models may improve accuracy but require more processing time.
- english: Enables the English-specific model for potentially higher accuracy on English audio.

- verbose: Provides detailed output about the transcription process for debugging purposes.
- Additional options include --energy, --dynamic energy, --pause, and --save file for further customization of the audio processing and saving behavior.

How It Works

The Live Transcription System operates by capturing audio through the microphone, processing the audio to extract speech, and then transcribing the speech into text. The system uses the speech recognition library to capture audio from the microphone. Here is a simplified code snippet demonstrating the audio capture process:

```
with sr.Microphone(sample_rate=16000) as source:
    audio = r.listen(source)
```

This code configures the microphone to listen for audio and captures the audio data for processing

Transcription with Whisper

Once audio is captured, it is transcribed using the Whisper model selected by the user. The transcription process involves converting the audio data into a format compatible with Whisper and then obtaining the transcription result:

```
result = audio
_model.transcribe(audio_data)
```

This snippet shows how the captured audio data is fed into the Whisper model to generate the transcription. The ‘transcribe‘ method of the Whisper model processes the audio and returns the transcribed text.

Merging Transcripts

After transcription, individual transcripts can be merged into a single document. This feature is especially useful for sessions with multiple audio inputs or for users who wish to compile

transcriptions from various sources. The merging process is handled automatically upon termination of the program:

```
def merge_transcripts():
    # Code to merge individual transcript files
```

This function collects all transcript files from a specified directory, concatenates their contents, and saves the combined text into a new file, creating a comprehensive transcript.

Advanced Configuration

For users requiring finer control over the transcription process, the system provides several options for customization.

Energy Threshold Adjust the microphone's sensitivity to sound

```
--energy 400
```

This option sets the energy threshold for the microphone, influencing its ability to distinguish speech from background noise.

Dynamic Energy Adjustment

Enable dynamic adjustment of the microphone's sensitivity:

```
--dynamic_energy True
```

When enabled, this feature allows the system to automatically adjust the energy threshold based on ambient noise levels, improving speech detection in varying environments.

Code Overview

An in-depth look at key components of the Live Transcription System:

Signal Handling

The system implements signal handling to ensure graceful termination and cleanup:

```
def signal_handler(sig, frame):
    global stop_threads
    stop_threads = True
    merge_transcripts()
    print("Merged transcripts successfully.")
    os._exit(0)
```

This code snippet shows how the system responds to an interrupt signal (e.g., CTRL+C), signaling threads to stop and triggering the merging of transcripts before exiting.

5 Troubleshooting & Support

5.1 Error Messages or Behaviors

The main issue with our voice recognition is when there is an abundance of background noise. The model gets confused if there is an abundance of noise that makes it difficult to hear the user. This will result in the model interpreting the users words incorrectly and often ignoring the words all together.

A secondary issue that will occur is when the user's pronunciation is not good. We tested this by having members of our team use the model in languages that weren't their native tongue. The model often accurately interprets the user's words but would sometimes invent words that sounded closer to what the user pronounced.

5.2 Special Considerations

As seen above the special considerations for when using this model is when the user is in a high noise environment or when they are using the model in a different language then their native tongue.

5.3 Maintenance

The website does not require any regular maintenance from the users end.

6 Product Documentation

6.1 Advanced Software Architecture

6.1.1 Front-End React.js Ecosystem

- Developed a robust React.js front-end to handle real-time audio processing and transcription visualization, designed to accommodate future integration with OpenAI's Whisper model.
- Architectural planning includes modular components to easily insert Whisper model APIs and data handling mechanisms.

6.1.2 Bulma CSS for Adaptive UI

- Adopted Bulma CSS framework for its flexibility and adaptive design, which is crucial for users with varied accessibility needs.
- UI components are constructed to be reusable and modifiable for future states, including Whisper model feedback and interaction.

6.2 Integration Readiness for Whisper

6.2.1 Current Speech Recognition Mechanics

- The app presently harnesses the Web Speech API, with structures in place for a seamless transition to Whisper's advanced speech-to-text capabilities.
- Designed to be agnostic to the speech recognition service, the transition will involve minimal disruption to existing functionalities.

6.2.2 Preparing for Whisper Error Handling

- Error handling frameworks are designed to be robust and flexible to manage potential inaccuracies or training feedback from Whisper.

6.3 Development Considerations for Data Handling

6.3.1 Tools and Libraries for Machine Learning

- Node.js environment includes dependencies for machine learning libraries essential for processing data for Whisper training.
- Provisioned for handling large datasets with efficient data streaming and batch processing techniques.

6.3.2 Quality Assurance and Accessibility Testing

- Implemented rigorous testing protocols focusing on accessibility to ensure the application meets the needs of speech-impaired users.
- Comprehensive test suites and manual user testing regimes in place to maintain high-quality user experiences.

6.4 UI/UX Design for Data Capture

6.4.1 Intuitive Data Collection Interface

- The application's interface includes intuitive controls for recording speech, reviewing transcriptions, and submitting data for model training.
- Designed UI flows encouraging user participation in Whisper model data, with clear consent and instruction.

6.4.2 Enhanced Accessibility Features

- Ensured high contrast, keyboard navigability, and screen reader compatibility to serve users with diverse accessibility requirements.

6.5 Performance Optimization for Data Volume

6.5.1 Efficient Data Management Strategies

- Developed a data management system to handle and preprocess audio data efficiently, ready for ingestion by the Whisper model training pipeline.
- Optimized database schema for fast retrieval and storage of transcriptions and user interactions.

6.5.2 Scalable Application Infrastructure

- Ensured the application backend could scale to meet the demands of data-intensive operations related to Whisper model training.

6.6 Deployment Strategy for Continuous Evolution

6.6.1 Cloud Solutions for High-Performance Computing

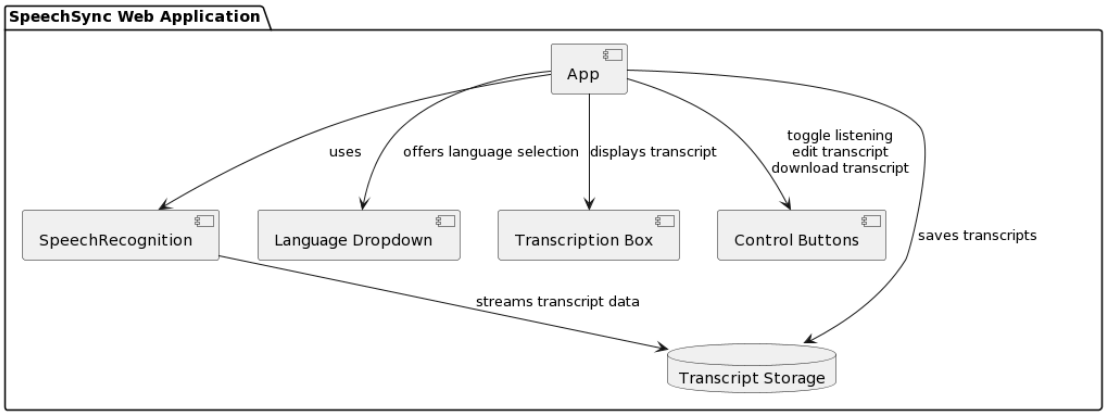
- Selected cloud solutions offering high-performance computing options to handle the resource-intensive nature of model training.
- The deployment strategy accounts for the need for rapid scaling and deployment of updated models.

6.6.2 CI/CD for Smooth Model Iteration Rollouts

- A CI/CD pipeline is in place, designed to handle frequent updates as the Whisper model undergoes iterative training and refinement.

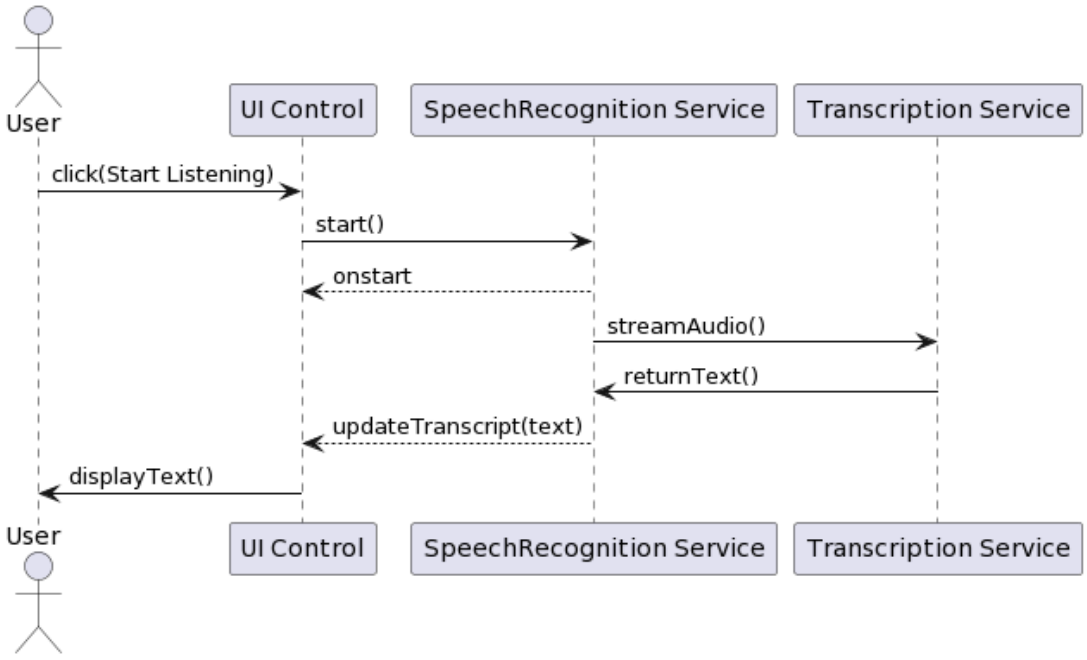
Supporting Documentation

6.7 Component Hierarchy and Data Flow Description



- The image describes the component hierarchy and data flow within the application is available for reference.

6.8 Sequence and Interaction Flows



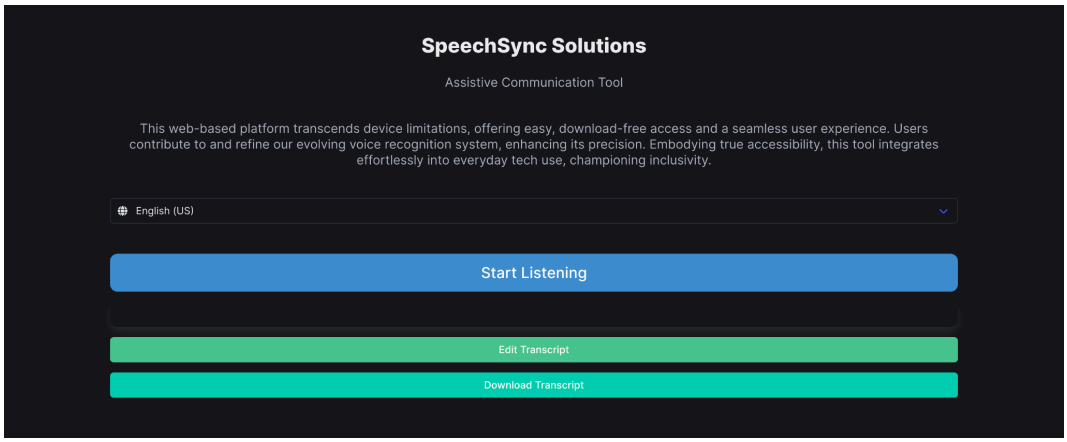
- Detailed descriptions of the sequence and interaction flows are provided above to depict how the user's actions trigger processes within the application.

6.9 API Documentation

- Comprehensive API documentation based on the Web Speech API is available at the Web Speech API Specification.

6.10 Code Repository

- The complete application code, including "App.js," is maintained in a version-controlled environment, accessible at GitHub - live-transcribe.



6.1 <Subsystem 1 of prototype>

6.1.1 BOM (Bill of Materials)

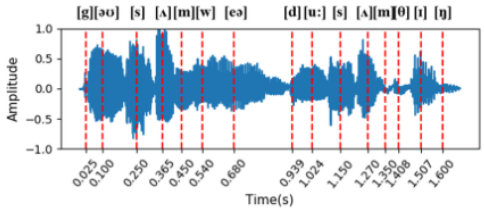
6.1.2 Equipment list

In the end our project did not use any physical materials and the end result was purely digital. After our initial idea of fabricating a button/switch that could be used to control the clients phone was scrapped so too was the initial BOM containing the necessary materials. We then fell back to our secondary plan of a voice recognition system that involved a physical microphone connected to a raspberry pi however due to time constraints we opted for a zero material website instead.

6.2 Testing & Validation

Before finalising the website we had a few tests done to make sure everything was working as intended, we first tested if the voice recognition system worked with our voices with normal speech, we then used the voice samples taken from our client and played them all to test the accuracy of the recognition, after a few adjustments to the python code, we fixed the accuracy to make it almost 90% accurate even with our clients slurred speech. After the client meetings, we knew that our clients main problem was that hey google was cutting off before the client finished his sentence, which is why after finishing the code, we added a start and stop listening button to solve that issue, it required minimal testing due to its simplicity. Our final test was to see how well our app works when in a public area and theres a lot of noise around us, the client wont be in a silent room all the time so this was an important test. After testing in different conditions, the loud voices of other people affected the apps accuracy, unless the speaker was really close to the

microphone. So we decided that a collar microphone or even a small microphone stand would do the trick and solve that issue easily.



The picture above for example represents the amplitude at different times and different sounds that the app recognizes after tests.

7 Conclusions and Recommendations for Future Work

Recommendations

Migrating the Live Transcription System to operate as a standalone device on a Raspberry Pi offers a portable and accessible solution for real-time speech-to-text transcription. This section outlines the steps for migration and enhancements tailored to assist speech-impaired individuals.

Migrating to Raspberry Pi

Hardware Requirements

• A Raspberry Pi 4 Model B or later for optimal performance.

• A USB microphone compatible with Raspberry Pi for audio input.

• An SD card with adequate storage for the OS, application, and data files.

Raspberry Pi Setup:

1. Install Raspberry Pi OS using the Raspberry Pi Imager tool and update the system with the commands: sudo apt-get update and sudo apt-get upgrade.

2. Ensure Python 3.8 or later is installed and install Pip if necessary.

3. Transfer the Python script to the Raspberry Pi and install required dependencies via Pip.

Autostart Configuration: Configure the script to execute automatically on boot by adding a cron job or modifying the .bashrc file.

Enhancing for Speech-Impaired Users

Custom Model Training: To improve accuracy for speech-impaired speech patterns, consider training a custom model with a dataset from speech-impaired individuals.

User Interface Improvements: Develop a user-friendly interface with support for touch screens, providing visual feedback and easy control over the transcription process.

Accessibility Features: Incorporate adjustable text sizes, high-contrast themes, and visual cues to accommodate users with various needs.

Real-Time Feedback: Enhance the system to provide real-time transcription feedback, allowing users to see and adjust their speech as needed.

Integration with Communication Aids: Explore possibilities for connecting the system with other communication tools used by speech-impaired individuals, such as text-to-speech devices.

Continuous User Feedback: Regularly update the system based on feedback from speech impaired users to better meet their communication needs

Conclusion

The transition of the Live Transcription System to a Raspberry Pi platform represents a significant step towards creating an accessible, cost-effective solution for real-time speech-to-text conversion. This migration not only enhances the system's portability and usability but also opens up new avenues for customizing the solution to better serve the needs of speech-impaired individuals. By focusing on this particular user group, the system becomes more inclusive, offering a means of communication that was previously more difficult or even inaccessible for some.

Further tailoring of the Live Transcription System for speech-impaired users, coupled with the potential for Raspberry Pi-based standalone operation, exemplifies the impactful synergy between cutting-edge technology and user-centric design. These efforts not only improve the system's adaptability and effectiveness across diverse user groups but also underscore the importance of accessibility in technology development.

Moreover, the Live Transcription System's reliance on Whisper models for audio transcription demonstrates the powerful capabilities of modern AI in interpreting and processing natural language. The system's customizable options and efficient processing mechanisms offer a valuable tool for a wide range of applications, from assisting individuals with speech impairments to facilitating the transcription needs of professionals across various fields.

As technology continues to advance, the opportunities for enhancing and expanding the Live Transcription System will grow. Collaborative development, driven by feedback from users and contributions from the wider community, will be key to realizing the full potential of this innovative tool. Your feedback and contributions are not only welcomed but essential for the continuous improvement of the system, ensuring it remains a relevant, powerful, and accessible

tool for converting speech to text.

APPENDICES

8 APPENDIX I: Design Files

https://live-transcribe.netlify.app/ is the app website