Mini-Project Report On

Speech-to-Text Summarizer Application using Transformer-based NLP Models

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

in

Computer Science & Engineering

By

Rohan Ranjith (U2003172)
S Gokul Raj (U2003179)
Sneha Sarah John (U2003203)
Varun Pradeep (U2003211)

Under the guidance of
Dr. Dhanya P M

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY
(AUTONOMOUS)

Department of Computer Science & Engineering
Rajagiri School of Engineering and Technology (Autonomous)
Rajagiri Valley, Kakkanad, Kochi, 682039

July 2023

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
RAJAGIRI SCHOOL OF ENGINEERING AND TECHNOLOGY
(AUTONOMOUS)
RAJAGIRI VALLEY, KAKKANAD, KOCHI, 682039

CERTIFICATE

This is to certify that the mini-project report entitled ***"Speech-to-Text Summarizer Application using Transformer-based NLP Models"*** *is a bonafide work done by* ***Mr. Rohan Ranjith (U2003172), Mr. S Gokul Raj (U2003179), Ms. Sneha Sarah John (U2003203), Mr. Varun Pradeep (U2003211)****, submitted to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (B. Tech.) in Computer Science and Engineering during the academic year 2022-2023.*

Dr. Preetha K. G.	**Ms. Anita John**	**Dr. Dhanya P M**
Head of Department	Mini-Project Coordinator	Mini-Project Guide
Dept. of CSE	Asst. Professor	Associate Professor
RSET	Dept. of CSE	Dept. of CSE
	RSET	RSET

ACKNOWLEDGEMENTS

We wish to express our sincere gratitude towards **Dr. P. S. Sreejith**, Principal of RSET, and **Dr. Preetha K. G.**, Head of Department of Computer Science and Engineering for providing us with the opportunity to undertake our mini-project, "Speech-to-Text Summarizer Application using Transformer-based NLP Models".

We are highly indebted to our mini-project coordinators, **Ms. Anita John**, Assistant Professor, Department of Computer Science and Engineering, and **Mr. Sajanraj T D**, Assistant Professor, Department of Computer Science and Engineering for their valuable support.

It is indeed our pleasure and a moment of satisfaction for us to express our sincere gratitude to our mini-project guide **Dr. Dhanya P M**, for her patience and all the priceless advice and wisdom she has shared with us.

Last but not the least, we would like to express our sincere gratitude towards all other teachers and friends for their continuous support and constructive ideas.

Rohan Ranjith

S Gokul Raj

Sneha Sarah John

Varun Pradeep

ABSTRACT

The goal of this project is to design and develop a comprehensive system capable of converting recorded speech into accurate textual representation and subsequently generating a concise summary of the transcribed text, focusing on the key points and main ideas expressed during the speech. This system aims to bridge the gap between spoken language and written content, enabling efficient information retrieval and comprehension of lengthy audio recordings

The objective of this project is to create an automated system that can summarize spoken language into concise written text with efficiency and accuracy. The system aims to save time and effort by condensing lengthy audio recordings into a coherent representation. By providing a summary, users can quickly grasp the main ideas and essential information expressed during the speech.

The system utilizes advanced natural language processing techniques to convert spoken language into text. It employs speech recognition algorithms to transcribe audio recordings accurately. The transcribed text is then processed further to extract the most relevant information. Efficiency is central to the system, automating the summarization process to eliminate manual listening or reading of lengthy audio recordings. Users receive a concise summary capturing the speech's essence, saving time and reducing the cognitive effort required for understanding.

Contents

List of Figures

Chapter 1

Introduction

Speech is an essential part of human communication enabling the exchange of information as well as fostering important connections. It plays a pivotal role in our information dense society, and could often contain details that do not contribute to the core message being conveyed. Hence there arises a need to condense speech and retrieve only the essential elements. "The original form of the speech is a signal, and a signal is processed such that all the information present in the signal is converted into the text format. Feature extraction is the process of taking a signal and converting it to the required format with certain logic."[5] This forms the basis of our project Sound Bite: a speech to text Summarizer.

1.1 Background

The following aspects highlight the necessity of inculcating a speech to text summarizer into our daily life:

Accessibility:

A speech-to-text summarizer can enhance accessibility for individuals with hearing impairments by converting spoken language into written text. It allows them to access information and participate in conversations or events that rely on verbal communication.

Efficiency and Time-saving:

Summarizing spoken language can save significant time and effort. In scenarios such as meetings, lectures, or conference calls, a speech-to-text summarizer[9] can automatically generate concise summaries, eliminating the need for manual note-taking or reviewing lengthy recordings.

Documenting and Archiving:

Organizations often need to document and archive meetings, interviews, or other spoken content. By transcribing and summarizing these recordings, important information is captured and preserved in a written format, making it easily accessible for future reference and analysis.

Language Learning and Education:

A speech-to-text summarizer can aid language learners by providing written transcripts and summaries of spoken content. This helps learners better understand and study the language, identify key vocabulary, and reinforce comprehension.

Streamlining Journalism Workflows:

Speech-to-text summarizers streamline workflows, provide easy reference to quotes, facilitate rapid content analysis, aid in fact-checking, enhance collaboration among journalists, and enable adaptive news consumption.

Text Preprocessing:

NLP techniques are used to preprocess the input text before summarization, including tokenization, stop word removal, and part-of-speech tagging.

Information Extraction:

NLP is used to extract important information from the text, such as named entity recognition and relation extraction.

Sentence Ranking and Selection:

NLP algorithms rank and select the most important sentences for inclusion in the summary, based on relevance, semantic similarity, or keywords.

Text Compression:

NLP techniques are employed to compress selected sentences or passages, such as sentence fusion or reduction.

Abstractive Summarization:

NLP techniques generate new sentences that capture the essence of the original text, using natural language generation techniques like language modeling and sequence-to-sequence models.[10]

Evaluation and Quality Assessment:

NLP is used to evaluate the quality and coherence of generated summaries, employing metrics like ROUGE and BLEU.[6]

Multilingual Summarization:

NLP enables summarization in multiple languages, utilizing machine translation and cross-lingual information retrieval techniques.

The utilization of NLP techniques in the summarization process allows for efficient text preprocessing, information extraction, sentence ranking, compression, abstractive generation, evaluation, and multilingual support. These techniques empower the development of effective and accurate summarization systems capable of processing and condensing large volumes of text into concise summaries.

1.2 Existing System

Journalists often rely on audio recording devices to capture interviews, but manually sifting through lengthy recordings to identify important information can be time-consuming.

Deaf individuals benefit from speech-to-text systems; however, these systems may generate text with unnecessary filler words that hinder efficient comprehension.

Traditional summarization models struggle to capture context and nuanced meanings, particularly when dealing with specialized jargon.

NLP algorithms and smaller models are commonly used for summarization, but their effectiveness is often limited to extractive methods or basic abstractive techniques[10], which may not provide substantial assistance.

1.3 Problem Statement

The problem addressed by an abstractive speech-to-text summarizer is the need for an automated method to summarize spoken content to replace individuals manually transcribing and condensing. Existing speech recognition systems predominantly focus on providing verbatim transcripts, lacking the ability to generate concise and coherent summaries that capture the essence of the original speech.

This system aims to bridge the gap between spoken language and written content, enabling efficient information retrieval and comprehension of lengthy audio recordings.

1.4 Objectives

The objective of an abstractive speech-to-text summarizer is to convert spoken language into concise and coherent summaries that capture the essence of the original speech, utilizing advanced natural language processing and deep learning techniques. It aims to generate summaries that go beyond mere extraction of key phrases, by understanding the context, generating new phrases, and maintaining overall coherence. The ultimate goal is to provide a human-like summarization capability, enabling efficient information retrieval and enhancing accessibility for individuals with hearing impairments or those seeking quick and accurate summaries of audio content.

1.5 Scope

T he scope of this topic encompasses various aspects, starting with the development of accurate speech recognition models[8] that can transcribe audio or video recordings. Preprocessing techniques such as tokenization and part-of-speech tagging are applied to the transcribed text to prepare it for the summarization process.

The next phase involves the design and implementation of summarization algorithms[10], which can be extractive or abstractive in nature. Extractive methods select important sentences or passages from the text, while abstractive methods generate new sentences that capture the main ideas. Evaluation metrics are used to assess the quality and coherence of the generated summaries[10]. Additionally, considerations are given to user interfaces and system performance to ensure efficient access to the summaries and real-time processing of audio input. Lastly, domain-specific challenges, ethical considerations, and documentation of methodologies and findings play crucial roles in advancing research and application in speech-to-text summarization.

Chapter 2

Literature Review

2.1 Summarization Based on BART

BART, short for "Bidirectional and Auto-Regressive Transformer," is a state-of-the-art pre-trained language model that belongs to the family of transformer models. It was introduced by Facebook AI Research in 2019. BART is known for its effectiveness in various natural language processing tasks, including text generation, text completion, summarization, and translation.

Unlike some other models that are trained in either a left-to-right (auto-regressive) or a right-to-left (auto-regressive) manner, BART combines both approaches. It leverages a bidirectional encoder-decoder architecture, enabling it to capture contextual information from both directions. This bidirectional capability contributes to BART's ability to understand and generate coherent and contextually relevant text.

BART's training process involves two main steps: pre-training and fine-tuning. During pre-training, the model is trained on a large corpus of text data using various self-supervised learning techniques. This includes tasks such as masked language modeling and denoising autoencoding, which help the model learn to predict missing or corrupted words in a sentence. In the fine-tuning phase, BART is further trained on specific downstream tasks, such as summarization or translation, to adapt its knowledge to more targeted applications.

BART has demonstrated impressive performance in various natural language processing tasks. In the context of summarization, BART can generate abstractive summaries by understanding the context and generating new sentences that capture the essence of the input text. Its ability to handle long-range dependencies and capture global context makes it particularly well-suited for summarization tasks.

BART has proven to be an effective and versatile model, advancing the capabilities of natural language processing and contributing to advancements in text generation and understanding.

BART, while being highly effective and versatile, does have certain drawbacks. It is computationally expensive to train and deploy, limiting its accessibility for resource-constrained applications. Fine-tuning BART can be challenging, requiring careful dataset curation and optimization of hyperparameters. The model's abstractive nature may result in less control over the generated output, potentially leading to inaccuracies or inconsistencies. BART's large size contributes to increased inference latency, making it less suitable for real-time applications. Biases present in the training data may be reflected in the generated summaries, and the model may struggle with out-of-distribution data. Interpretability can also be a challenge as the decision-making process is not easily explainable. Considering these limitations is crucial when utilizing BART in specific contexts and evaluating its performance.[7]

2.2 Summarization based on T5

T5, which stands for "Text-To-Text Transfer Transformer," is a cutting-edge language model introduced by Google AI in 2019. T5 is known for its versatility and ability to perform a wide range of natural language processing tasks by transforming the input text into a text-to-text format. It is based on the transformer architecture and has achieved remarkable results in various tasks, including text classification, translation, summarization, and question-answering.

The distinguishing feature of T5 is its unified framework, where all tasks are cast as text-to-text transformations. This means that both input and output are represented as text strings, allowing T5 to be trained in a consistent manner across different tasks. By conditioning the model on the desired input-output format, T5 can be fine-tuned for specific tasks while benefiting from the knowledge acquired during pre-training.

T5's versatility and strong performance are attributed to its large-scale pre-training using diverse and extensive datasets. This pre-training phase involves training the model on a massive amount of text data, enabling it to learn rich language representations. Fine-tuning T5 on task-specific data further refines its performance on targeted applications.

The effectiveness of T5 has led to its widespread adoption and significant contributions to advancements in natural language processing research and applications.

While T5 is a powerful and versatile language model, it does have certain drawbacks worth considering. Firstly, the training and fine-tuning of T5 can be computationally intensive and resource-demanding, requiring significant computational power and time. This can limit its accessibility for users with limited resources or hinder real-time deployment in certain applications. Additionally, the large-scale pre-training of T5 on diverse datasets can introduce biases present in the training data, potentially affecting the fairness and objectivity of its generated outputs. Furthermore, T5's fine-tuning process often requires task-specific datasets, which may not be readily available or require substantial effort for curation. Finally, T5's sheer size and complexity can result in increased inference latency, making it less suitable for time-sensitive applications where real-time responses are required. These drawbacks highlight the need for careful consideration and trade-offs when utilizing T5 in specific contexts or applications. Ongoing research aims to address these limitations and further enhance the efficiency, fairness, and usability of models like T5.

2.3 Summarization model based on Pegasus

Pegasus is an advanced sequence-to-sequence model introduced by Google Research in 2020. It is specifically designed for abstractive text summarization, aiming to generate coherent and concise summaries from longer source documents. Pegasus builds upon the transformer architecture and leverages techniques such as pre-training and fine-tuning to achieve state-of-the-art performance in summarization tasks.

One of the key features of Pegasus is its utilization of a combination of unsupervised and supervised learning. During pre-training, Pegasus is trained on a massive corpus of publicly available text from the internet, enabling it to learn rich representations of language.

This is followed by fine-tuning on supervised summarization datasets, where it learns to generate abstractive summaries by mapping the source document to a shorter summary. Pegasus excels in handling long-range dependencies, maintaining the context, and generating coherent summaries with fluent language.

The success of Pegasus has been demonstrated in various benchmark datasets and competitions. It has surpassed previous state-of-the-art models in terms of both ROUGE scores and human evaluation. Pegasus has been widely adopted in research and practical applications where abstractive text summarization is required, providing a powerful tool for extracting key information and generating concise summaries from large bodies of text.

While Pegasus is a highly effective model for abstractive text summarization, it does have a few limitations to consider. Firstly, Pegasus relies heavily on large-scale pre-training and fine-tuning processes, which can be computationally demanding and time-consuming. This makes it challenging for users with limited resources to train or fine-tune the model effectively. Additionally, as with other language models, Pegasus may produce summaries that exhibit biases present in the training data, potentially resulting in biased or subjective outputs. Moreover, generating abstractive summaries requires a significant amount of training data with human-authored summaries, making it necessary to curate or create large and diverse datasets for fine-tuning. Lastly, similar to other sophisticated models, Pegasus can have high inference latency due to its size and complexity, making it less suitable for real-time or latency-sensitive applications. Addressing these limitations will contribute to further advancements and broader accessibility of abstractive text summarization techniques.[10]

2.4 Dragon NaturallySpeaking : A speech recognition software

Dragon NaturallySpeaking, developed by Nuance Communications, is a leading speech recognition software that converts spoken language into written text. It is designed to enhance productivity and accessibility by allowing users to dictate documents, emails, or perform various computer tasks using voice commands. Dragon NaturallySpeaking utilizes advanced acoustic and language models to accurately transcribe spoken words into text with high accuracy and speed.

The software employs deep learning and machine learning techniques to continuously adapt and improve its speech recognition capabilities. It can learn from user corrections and adjustments, gradually enhancing its accuracy and understanding of individual user's speech patterns.

Dragon NaturallySpeaking supports multiple languages and offers a wide range of features, such as voice commands for navigating applications, controlling the computer, and executing tasks hands-free.

With its robust speech recognition technology, Dragon NaturallySpeaking has found applications in various fields. It enables individuals with mobility impairments, repetitive stress injuries, or other physical disabilities to interact with computers more efficiently. It is also widely used by professionals, such as writers, researchers, and professionals in healthcare and legal industries, to increase productivity and streamline their workflow through voice-based input. Dragon NaturallySpeaking's continuous improvements in accuracy and its extensive feature set make it a popular choice for users seeking reliable and convenient speech recognition software.

One drawback of Dragon NaturallySpeaking is that it requires initial training and adaptation to individual user speech patterns, which can be time-consuming. Users may need to invest significant effort in training the software to improve accuracy and achieve optimal performance. Additionally, Dragon NaturallySpeaking's accuracy can be affected by ambient noise, varying accents, or speech impediments, requiring users to speak clearly and in a controlled environment for optimal results. The software's performance may also be influenced by the available computing resources, and it may require a powerful computer to operate smoothly.

2.5 Hidden Markov Model for speech recognition

A Hidden Markov Model (HMM) is a statistical model used to describe and analyze sequential data with underlying hidden states. It is based on the concept of Markov processes, where the current state depends only on the previous state. HMMs are commonly applied in various fields, including speech recognition, natural language processing, bioinformatics, and signal processing.

HMMs are utilized to capture the temporal dependencies and variability in speech patterns. The basic idea is to represent the speech signal as a sequence of hidden states, where each state corresponds to a particular phoneme or sub-phonetic unit.

HMMs in speech recognition consist of three main components: the set of hidden states, the transition probabilities between states, and the emission probabilities representing the acoustic properties of each state. These emission probabilities are typically modeled using Gaussian Mixture Models (GMMs) or Deep Neural Networks (DNNs). The HMM model allows for the estimation of the most likely sequence of hidden states given the observed speech signal using algorithms like the Viterbi algorithm.

By modeling speech as a sequence of hidden states and leveraging the transitional and emission probabilities, HMMs can effectively capture the dynamics and variability of speech. This makes them a foundational component in many state-of-the-art speech recognition systems, enabling accurate and robust speech-to-text conversion. HMM-based approaches have been successfully applied in various speech recognition applications, including dictation systems, voice assistants, and automatic speech recognition in domains such as healthcare, telecommunications, and transcription services.

One drawback of Hidden Markov Models (HMMs) is their assumption of a fixed and finite number of hidden states with predefined transitions. This assumption limits the flexibility of the model in capturing complex and dynamic relationships present in the data. HMMs may struggle to accurately represent long-range dependencies or handle situations where the number of states or transitions varies significantly. Additionally, HMMs have difficulty modeling simultaneous or overlapping events, as the model assumes a sequential nature of data. The performance of HMMs in speech recognition can be affected by variations in speech rate, speaker characteristics, or environmental noise, and they may struggle to adapt to these variations without additional techniques or modifications[6]. Despite these limitations, HMMs have been widely used and have served as a foundation for various advancements in speech recognition; however, more advanced models have emerged that overcome some of the limitations associated with HMMs.

Comparison

	BART	T5	Pegasus
Year	Introduced in 2019	Introduced in 2019	Introduced in 2020
Model Type	Seq2Seq model	Seq2Seq model	Seq2Seq model
Pre-training	Masked language modeling, denoising autoencoding	Text-to-text transfer learning	Unsupervised pre-training, supervised fine-tuning
Task Focus	Text generation, summarization, translation, question-answering	Wide range of NLP tasks, including summarization, translation, classification	Abstractive text summarization
Architecture	Encoder-Decoder transformer	Encoder-Decoder transformer	Encoder-Decoder transformer
Fine-tuning	Task-specific fine-tuning on downstream tasks	Task-specific fine-tuning on downstream tasks	Supervised fine-tuning on summarization tasks
Performance	High performance in various NLP tasks, strong abstractive summarization capabilities	Versatile and strong performance in multiple NLP tasks	State-of-the-art performance in abstractive summarization
Data Needs	Large-scale diverse datasets required for pre-training and fine-tuning	Large-scale paired text-audio data for training and fine-tuning	Pre-training on large corpus, fine-tuning on supervised summarization datasets
Model Size	Large model size	Large model size	Large model size
Real-time Inference	Inference latency may be a concern	Inference latency may be a concern	Inference latency may be a concern
Interpretability	Interpretability challenges	Interpretability challenges	Interpretability challenges
Biases	Potential biases from training data	Potential biases from training data	Potential biases from training data

Figure 2.1: Comparison of existing summarization models

	Dragon NaturallySpeaking	Hidden Markov Models (HMM)
Approach	Commercial speech recognition software	Statistical model for speech recognition
Language Modeling	Utilizes acoustic and language models	Uses transitional and emission probabilities
Training and Adaptation	Requires initial training and user adaptation	Trained on large speech datasets
Adaptability	Learns from user corrections and adjustments	Limited adaptability without retraining
Noise Sensitivity	Affected by ambient noise, requires controlled environment for optimal results	Performance affected by variations in speech rate, noise, or speaker characteristics
Vocabulary	Supports a wide range of vocabulary and language models	Vocabulary depends on training data and modeling choices
Real-time Interaction	Enables real-time speech-to-text conversion	Real-time interaction possible with fast algorithms and model optimization
Flexibility	Relies on pre-defined rules and patterns	Captures sequential dependencies, limited flexibility in capturing dynamic relationships
Computational Resources	Requires a powerful computer for smooth operation	Relatively less resource-intensive
Application Range	Widely used for productivity and accessibility, professional use in various industries	Applied in various fields including speech recognition, bioinformatics, and signal processing
Accuracy	High accuracy in speech-to-text conversion	Performance varies depending on the quality of training data and modeling choices
Language Coverage	Supports multiple languages	Language-dependent, can be tailored to specific languages
Inference Latency	Low inference latency, near real-time response	Not a primary concern, typically offline analysis

Figure 2.2: Comparison of existing speech recognition models

Chapter 3

System Analysis

3.1 Expected System Requirements

The system of user which is a computer is expected to have the following features:

- Any latest version PC platform (Windows,Mac OS,Linux).
- Requirement of Internet connection
- A minimum Ram size of 4GB is required in the device.

3.2 Feasibility Analysis

3.2.1 Technical Feasibility

The project is technically feasible since the majority of the population are in possession of a computer. The application requires only a web browser, and a microphone to run.

3.2.2 Operational Feasibility

The operations are built in a simple and easy to use manner for people with different needs.

3.2.3 Economic Feasibility

The app can reduce the expense incurred by people in order to maintain stationery such as notebooks, files, registers etc. The development of the application is also zero budget as it was built using free resources. Only minimal operational costs from the use of the chatGPT API which provides API access at low cost(0.002$ for 1000 tokens)[2].

3.3 Hardware Requirements

The following are the system requirements to develop the Sound-Bite App.

- Processor: Intel Core i5
- Hard Disk: Minimum 128GB
- RAM: Minimum 8GB

3.4 Software Requirements

The following is the software used in the development of the app.

Operating System: Windows

3.4.1 Visual Studio Code for app development

Visual Studio Code is a popular and powerful source code editor developed by Microsoft. It provides a lightweight yet feature-rich environment for coding across multiple programming languages. With its intuitive user interface, extensive customization options, and a vast array of extensions, Visual Studio Code offers a highly flexible and personalized coding experience.

It supports a wide range of features, including syntax highlighting, code completion, debugging tools, version control integration, and intelligent code navigation. The editor's integrated terminal allows developers to run commands and scripts without leaving the editor.

3.4.2 Flask for web framework

Flask is a lightweight web framework for Python that offers simplicity and flexibility. It follows a micro-framework approach, providing essential features for web development without unnecessary complexity.[1]

With Flask, developers can build everything from simple websites to complex APIs. The framework offers extensions for integrating functionalities like database management and authentication.

Flask uses a straightforward routing system, making it easy to handle different HTTP methods and parameters. It also includes a built-in development server for convenient testing and debugging.

3.4.3 MongoDB for database

MongoDB is a popular NoSQL document database that offers high scalability and flexibility. It stores data in flexible, JSON-like documents, allowing for easy schema evolution and agile development.

MongoDB supports automatic sharding, enabling horizontal scaling across multiple servers to handle large amounts of data. It provides powerful querying capabilities and supports a rich set of operations, including indexing, aggregation, and geospatial queries.

MongoDB's flexible data model and dynamic schema make it suitable for a wide range of applications, including real-time analytics, content management systems, and mobile apps. The database offers built-in replication and failover mechanisms, ensuring high availability and data durability.

3.4.4 HTML, CSS, JS for UI development

HTML, CSS, and JavaScript are the fundamental technologies for building web pages and web applications.

HTML (Hypertext Markup Language) provides the structure and content of web pages, defining elements such as headings, paragraphs, links, and images.

CSS (Cascading Style Sheets) is used to define the visual presentation and layout of web pages, allowing developers to customize the colors, fonts, spacing, and positioning of HTML elements.

JavaScript is a versatile programming language that adds interactivity and dynamic behavior to web pages. It allows developers to handle user interactions, manipulate and update the HTML and CSS elements, and communicate with servers to fetch or send data asynchronously.

The combination of HTML, CSS, and JavaScript enables the creation of interactive and visually appealing web pages. HTML defines the structure, CSS enhances the appearance, and JavaScript adds functionality to respond to user actions and create dynamic experiences.

3.4.5 Google Colab for testing models

Running models in Google Colab is a seamless process that takes advantage of the platform's powerful computational resources. With Colab, you can write and execute code directly in the browser, eliminating the need for local machine setup. Whether it's training machine learning models, running data analysis, or executing deep learning algorithms, Colab provides an efficient environment to work with.

Colab supports popular libraries and frameworks such as TensorFlow, PyTorch, and scikit-learn, allowing you to import and utilize them in your code. You can install additional libraries using pip or conda commands. Colab also provides access to GPUs and TPUs, which can significantly speed up model training and inference for computationally intensive tasks.

To run a model in Colab, you simply need to write the code in the notebook cells, execute the cells, and observe the output. Colab offers features like syntax highlighting, code completion, and error checking to aid in the development process. You can also visualize and analyze your model's performance by plotting graphs and displaying results within the notebook itself.

Chapter 4

Methodology

4.1 Proposed Method

- Develop an application that can convert speech to text and summarize it.
- We use Whisper from OpenAI for transcription(speech to text) and ChatGPT also from OpenAI for summarization
- User gives audio as input and the application gives the summarized transcript as output.
- Application also contains features like template selection and summary search.

4.1.1 Summarization

ChatGPT and GPT-3.5 are state-of-the-art language models developed by OpenAI. These models are built upon the GPT (Generative Pre-trained Transformer) architecture and have been trained on massive amounts of text data to learn patterns, and context, and generate human-like responses.

ChatGPT is a variant of the GPT model that has been fine-tuned specifically for conversational interactions. It has been trained on dialogue data to generate more contextually relevant and engaging responses in a chat-based format. The training process involves optimizing the model to understand and generate appropriate replies based on the given conversation history. ChatGPT excels in generating coherent and contextually appropriate responses, making it a valuable tool for chatbots, virtual assistants, and other conversational applications.

GPT-3.5, on the other hand, is a more powerful and versatile language model. It has been trained on a vast corpus of internet text, including books, articles, and websites. GPT-3.5 can generate coherent and contextually relevant text across a wide range of tasks and prompts. It can perform tasks such as language translation, text completion, question answering, summarization, and much more. With millions (or even billions) of parameters, GPT-3.5 has an extensive knowledge base and can generate high-quality text outputs.[9]

Both ChatGPT and GPT-3.5 leverage the transformer architecture, which allows them to capture long-range dependencies and generate text based on the given context. These models are powered by deep learning techniques and can learn complex patterns and structures in the training data. However, it's important to note that while these models can generate impressive responses, they still have limitations and can occasionally produce incorrect or nonsensical outputs.

4.1.2 Transcription

OpenAI Whisper is a tool created by OpenAI that can understand and transcribe spoken language. This kind of tool is often referred to as an automatic speech recognition (ASR) system.

The way OpenAI Whisper works is a bit like a translator. It uses an encoder-decoder Transformer architecture. The 'encoder' understands the spoken language (the audio), and the 'decoder' generates the written text.[8]

To learn how to understand and transcribe spoken language, Whisper was trained using a huge amount of data from the internet - equivalent to continuously listening for over 77 years. This data is multilingual (includes many different languages) and multitask (covers different types of tasks, not just transcription).

The audio data that OpenAI Whisper learns from is processed in a specific way to make it easier for the system to understand. The audio is 're-sampled' to a standard quality (16,000 Hz, which is a measure of sound frequency), and then it's transformed into a visual representation (the '80-channel log-magnitude Mel spectrogram') that the system can learn from.

This transformation is done in small chunks (25-millisecond windows) that slightly overlap (a 'stride' of 10 milliseconds) to ensure no part of the audio is missed.

Chapter 5

System Design

5.1 Architecture Diagram

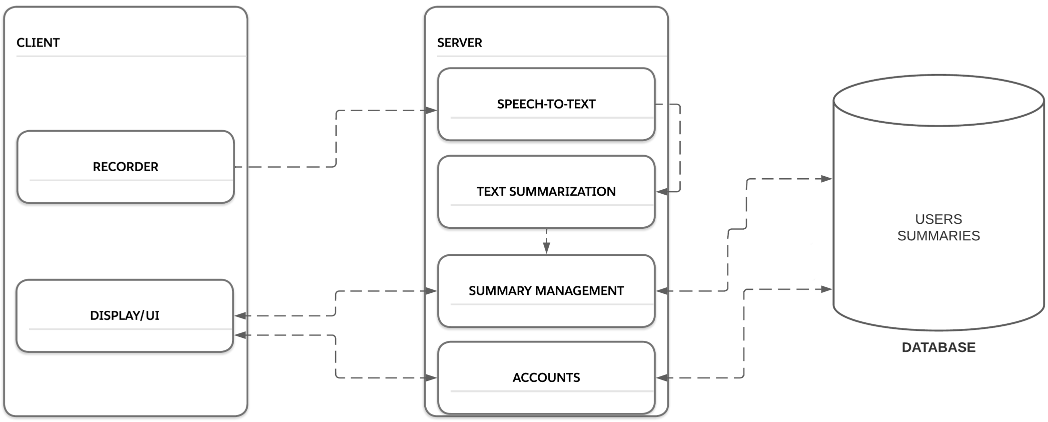

Figure 5.1: Architecture diagram

The Architecture diagram showcases the client-server architecture of the web application. It consists of three main sections: Client, Server, and Database.

The Client section includes the Recorder and Display modules, responsible for audio recording and presentation of data to the user.

The Server section incorporates several modules such as Speech to Text, Text Summarization, Summary Management, and Accounts. These modules handle the processing of recorded audio, transcription, summary generation, and user authentication.

The Database section comprises two collections: Users and Summaries. The Users collection stores user information, while the Summaries collection stores the generated summaries associated with their respective users.

5.2 Component Diagram

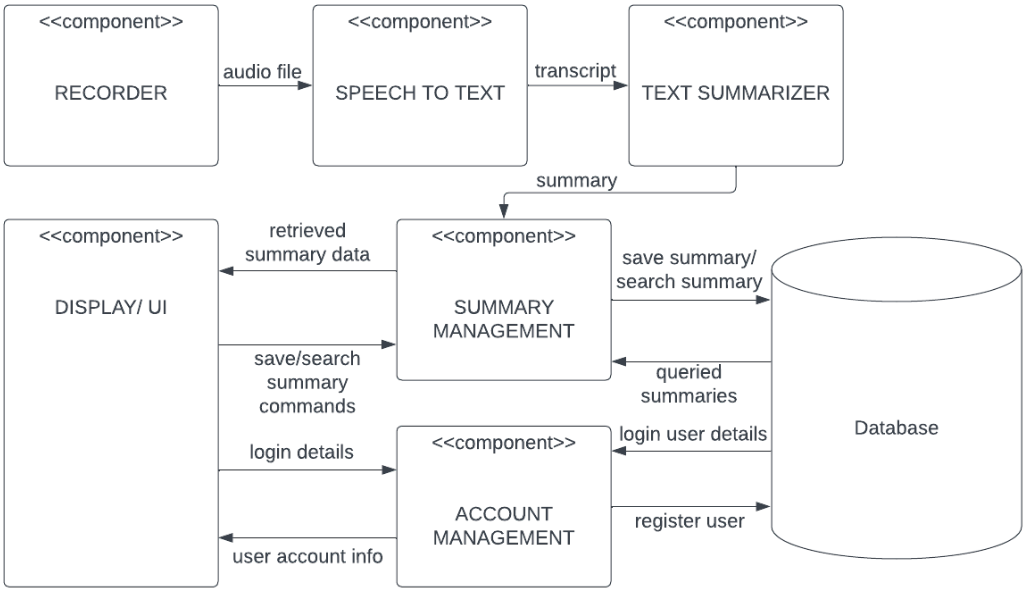

Figure 5.2: Component diagram

RECORDER: The Recorder module is responsible for capturing and storing the audio submitted by the user as input data to be processed further.[3]

SPEECH TO TEXT: Also known as the Transcriber, this module converts the audio file into a textual format, referred to as the transcription.[4]

TEXT SUMMARIZER: The Text Summarizer module simplifies the transcription by condensing it into its core details, producing a shortened version known as the summary.[2]

SUMMARY MANAGEMENT: Allows users to customize the summary's structure, add titles and categories, creating unique summaries that can be easily searched and managed in the database.

ACCOUNT MANAGEMENT: Handles user login, registration, logout, and associated session management. It ensures that only authenticated users can access protected routes.

DATABASE: The Database is the central repository where account details and summaries of each specific user are stored securely.

DISPLAY/UI: Represents the user interface through which users interact with the application, providing a seamless experience for audio recording, summary generation, and account management.

5.3 Use Case Diagram

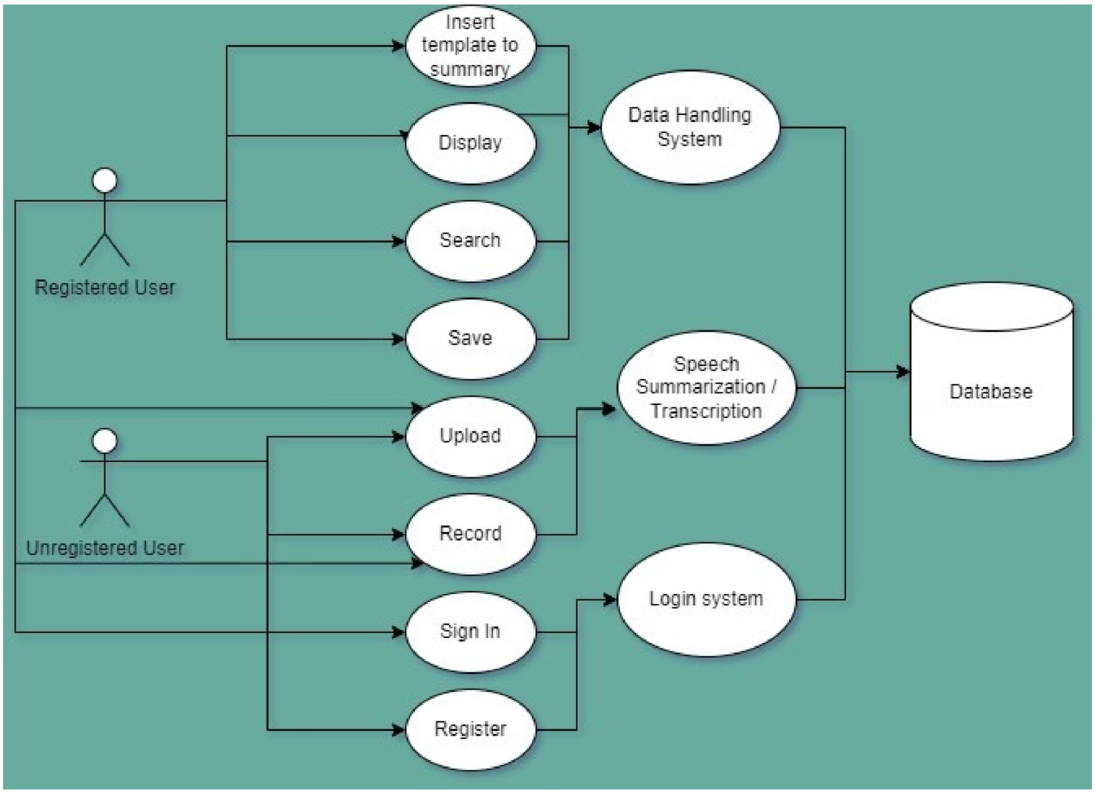

Figure 5.3: Use Case diagram

Actors include "Guest" and "Registered User" (authenticated users).

- **Guest:**
  - Upload Audio: Submit audio files for transcription and summary generation.
  - Record Audio: Capture and process audio for summary generation.
  - Register: Create a new account to access additional features.
- **Registered User:**
  - Fit Summary to Template: Customize summary format using various templates.
  - Display Summary: View generated summaries.
  - Save Summary: Store generated summaries for future reference.
  - Search Summary: Search for specific summaries based on keywords and categories.

5.4 Sequence diagram

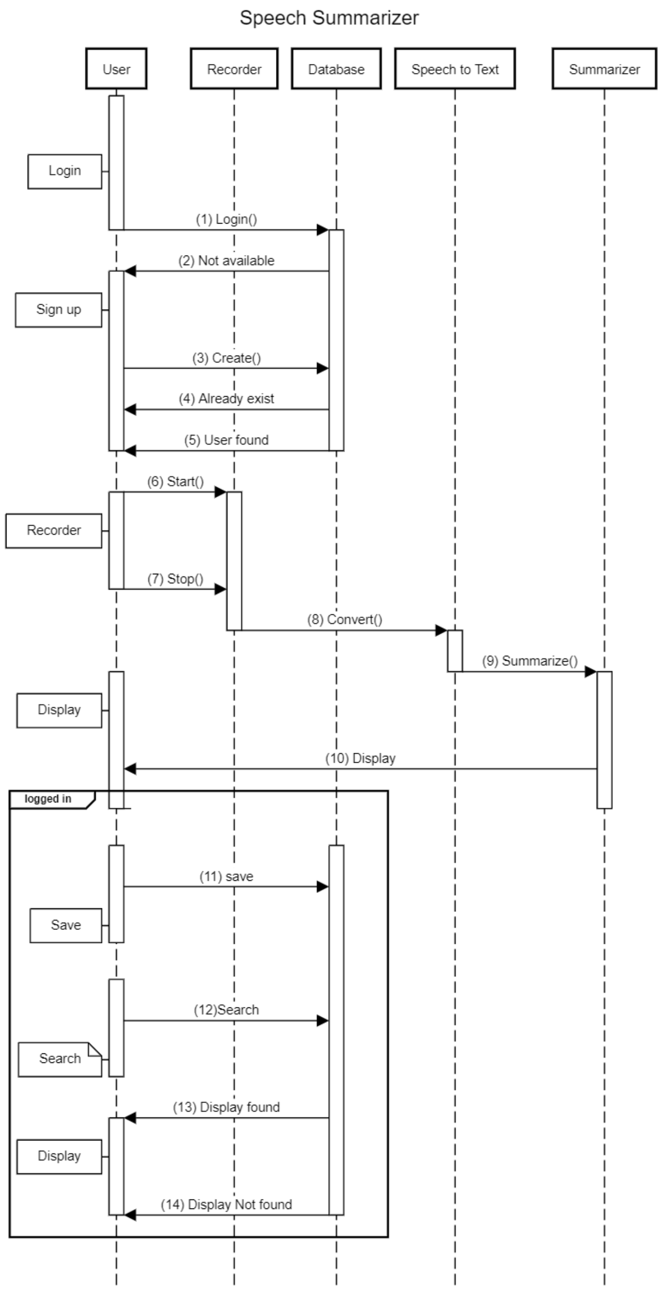

Figure 5.4: Sequence Diagram

Interactions, order of messages exchanged between different system components, objects.

Chapter 6

System Implementation

6.1 Recorder Module

- **Usage:** The Recorder module in the front-end handles audio recording functionality using a JavaScript recorder library[3]. It allows users to record audio, pause/resume recording, and stop recording. After stopping recording, the module provides a preview of the recorded audio, along with options to download the audio file and upload it to the server.

- **Implementation:** The front-end implements a user interface with three buttons: Record, Pause, and Stop. When the Record button is clicked, the recorder library starts recording audio. The Pause button allows the user to pause and resume recording without stopping the session. The Stop button finalizes the recording session. Upon stopping, the front-end appends a preview of the recorded audio to a list of audio files recorded in that session.

The preview typically includes an HTML `<audio>` element that allows users to play the recorded audio. Additionally, the preview may have a Download button, enabling users to download the audio file to their system. An Upload button is also provided, which triggers a call to the server's `/upload` API via a POST request. The recorded audio file is included in the request payload, which is sent to the server for further processing, including transcription and summary generation.

The Recorder Module in the front-end facilitates audio recording using a JavaScript recorder library. It provides controls for recording, pausing, and stopping audio capture. After stopping, it presents a preview of the recorded audio with options to download or upload the audio file. The upload functionality triggers the `/upload` API on the server, allowing further processing of the audio file, such as transcription and summary generation.

6.2 Accounts Module

- **Usage:** The Accounts module handles user authentication, registration, and session management functionalities. It ensures that only authorized users can access certain routes and APIs.

- **Implementation:** The module utilizes the Flask-Login library, a Python extension for managing user sessions and route protection. It provides the necessary functionality for login, registration, and logout.

6.2.1 Login Functionality

- **Usage:** The login functionality allows users to authenticate themselves using their email and password.

- **Implementation:** The front-end provides a simple user interface with input fields for email and password. When the user fills in their credentials and clicks the Login button, a POST request is sent to the `/login` API on the server. The server utilizes the Flask-Login library to validate the provided credentials against the stored user information in the MongoDB database. If the credentials are valid, the user is logged in and authenticated, allowing them access to protected routes and features.

6.2.2 Registration Functionality

- **Usage:** The registration functionality allows new users to create an account by providing their username, email, and password.

- **Implementation:** The front-end presents a user interface with input fields for username, email, and password. When the user enters their details and clicks the Register button, a POST request is sent to the `/register` API on the server. The server checks if the provided email is already registered. If not, it uses the Flask-Login library to hash the password, creates a new Subsequently, the user can log in with their registered email and password.

6.2.3 Logout Functionality

- **Usage:** The logout functionality allows authenticated users to end their session and log out.

- **Implementation:** The front-end includes a logout button in the navbar, visible to logged-in users. When the user clicks the logout button, it triggers a request to the `/logout` API on the server. The server utilizes the Flask-Login library to clear the user's session, effectively logging them out. After successful logout, the user is redirected to the home page or a designated logout page.

The Accounts Module encompasses three submodules: Login, Register, and Logout. The front-end provides user interfaces for inputting login credentials and registration details. Upon submission, the data is sent to the server's respective APIs for processing. The server utilizes the Flask-Login library to handle session management, authentication, and route protection. Successful login allows access to protected routes, while logout terminates the user's session.

6.3 Speech to Text Module

- **Usage:** The Speech to Text module converts the recorded speech into text by utilizing a model called Whisper[4], provided by OpenAI. The conversion is performed on the server side.

- **Implementation:** The module incorporates the Whisper model, which is an automatic speech recognition (ASR) system. The Whisper model runs locally on the server, allowing efficient and accurate transcription of the recorded speech. When the audio file is received through the `/upload` API, the server utilizes the Whisper model to transcribe the speech and convert it into textual form. This transcribed text is then passed to the Summary Management module for further processing.

The Speech to Text Module makes use of the Whisper model, provided by OpenAI, to perform the transcription of recorded speech into text. The model is deployed and runs locally on the server, ensuring efficient and accurate conversion. The audio file captured during the recording process is sent to the server via the `/upload` API, and the server uses the Whisper model to process the audio and generate the transcribed text. This text is subsequently utilized by the Summary Management module for summary generation and further processing.

6.4 Summary Management Module

The Summary Management Module consists of several submodules that handle different aspects of summary generation, saving, searching, and formatting.

6.4.1 Summarizer

- **Usage:** The Summarizer submodule utilizes OpenAI's ChatGPT model to generate a summary of the transcribed text.
- **Implementation:** When the `/upload` API receives the transcribed text, the Summarizer submodule makes use of the ChatGPT model to generate a concise summary. The generated summary is then returned to the front-end for display.

6.4.2 Save Summary

- **Usage:** The Save Summary submodule allows users to save the generated summary along with additional information, including the title, categories, timestamp, and email of the logged-in user.
- **Implementation:** The front-end provides an interface on the Save page where users can input the title, select categories, and save the generated summary. When the `/saveupload` API is called, the Save Summary submodule stores the summary, title, categories, timestamp, and user email in the MongoDB database, associating them with the logged-in user.

6.4.3 Search Summary

- **Usage:** The Search Summary submodule enables users to search for summaries based on specified search criteria, such as a substring in the title or summary body, categories, and date (by checking the timestamp).
- **Implementation:** The front-end provides an interface

for users to input search parameters, and when the `/search` API is called, the Search Summary submodule queries the MongoDB database for summaries that match the given criteria. The matching summaries are then returned to the front-end, where they are listed for the user.

6.4.4 Template Summary

- **Usage:** The Template Summary submodule converts the saved summary into a specified format, such as bullet points or minutes of a meeting, based on user preferences.

- **Implementation:** The front-end provides options for selecting a template format. When the `/template` API is called, the Template Summary submodule takes the saved summary and utilizes OpenAI's ChatGPT model to convert it into the specified format. The formatted summary is then returned to the front-end for preview.

APIs used:

- `/upload`: Receives the transcribed text and generates a summary using the Summarizer submodule.

- `/saveupload`: Saves the generated summary, title, categories, timestamp, and user email using the Save Summary submodule.

- `/search`: Queries the database for summaries based on search criteria using the Search Summary submodule.

- `/template`: Converts the saved summary into a specified format using the Template Summary submodule.

The Summary Management Module provides functionalities for summarizing the transcribed text, saving the summary with associated information, searching for summaries based on criteria, and formatting the summary into different templates.

6.5 Database Module

6.5.1 User Model

- **Attributes:**
  - `_id`: A unique identifier for the user document in the MongoDB collection.
  - `email`: The email address of the user.
  - `username`: The username chosen by the user.
  - `password`: The hashed password of the user. It is stored as a binary value.

6.5.2 Summary Model

- **Attributes:**
  - `_id`: A unique identifier for the summary document in the MongoDB collection.
  - `email`: The email address of the user who created the summary.
  - `title`: The title or name of the summary.
  - `categories`: An array of categories or tags associated with the summary.
  - `summary`: The content of the summary itself, providing a concise representation of the original text.
  - `timestamp`: A timestamp indicating the date and time when the summary was created. It is stored as a long integer.

6.5.3 Database

- **MongoDB Atlas:** The database used for storing the user and summary collections is MongoDB Atlas, a cloud-based database service provided by MongoDB. It allows for easy deployment, scalability, and management of the MongoDB database.

6.5.4 Collections

- **users:** This collection stores the user documents. Each document represents a registered user and contains attributes such as email, username, and hashed password.

- **summaries:** This collection stores the summary documents. Each document represents a summary created by a user and includes attributes such as email, title, categories, summary content, and timestamp.

The Database Module utilizes MongoDB Atlas as the database provider and includes two collections: `users` and `summaries`. The `users` collection stores user information, while the `summaries` collection stores summary documents associated with their respective users.

Chapter 7

Results

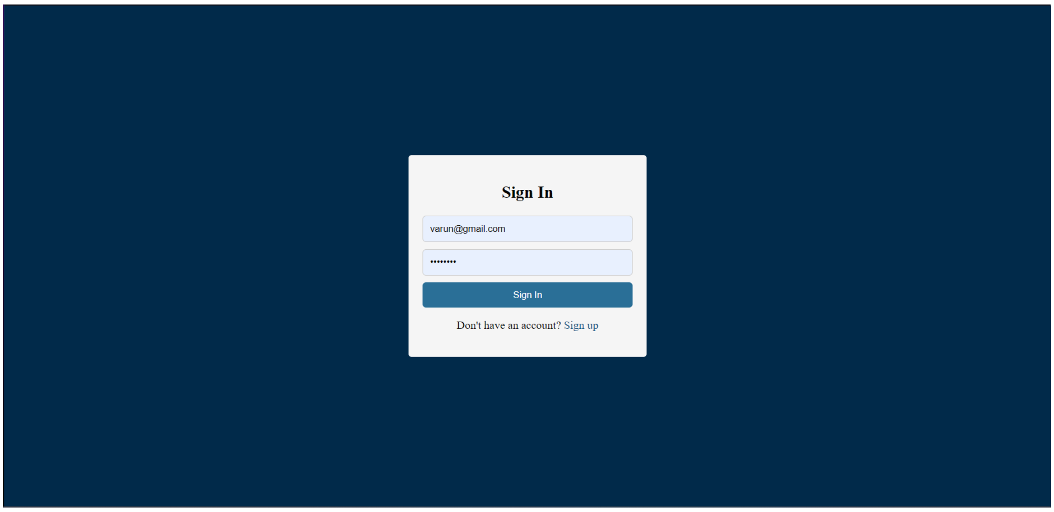

Figure 7.1: Login page

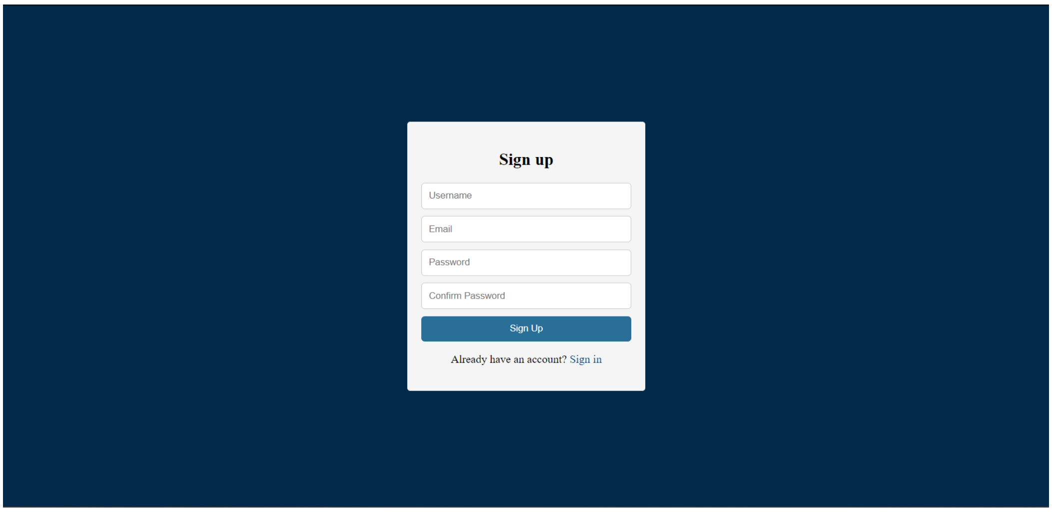

Figure 7.2: Register page

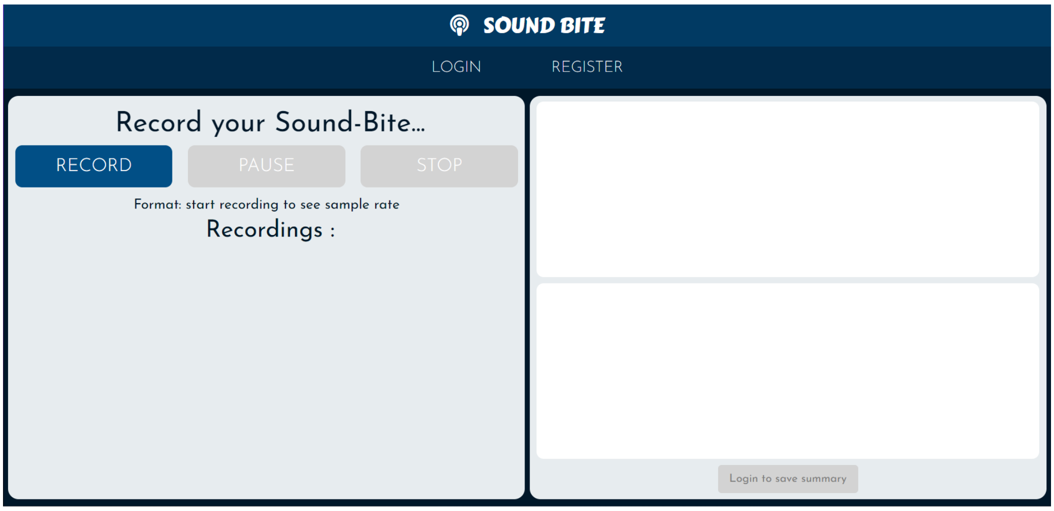

Figure 7.3: Home page - no login

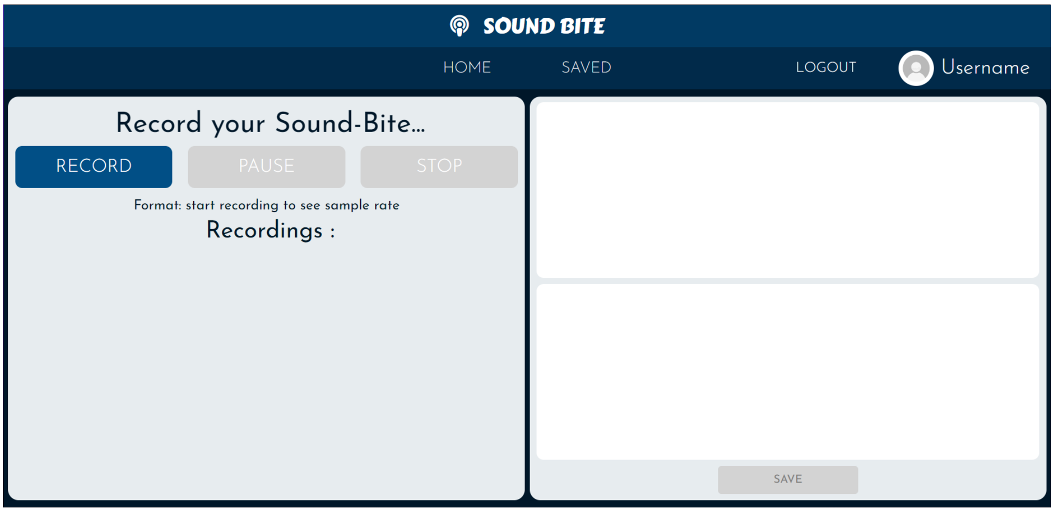

Figure 7.4: Home page - login

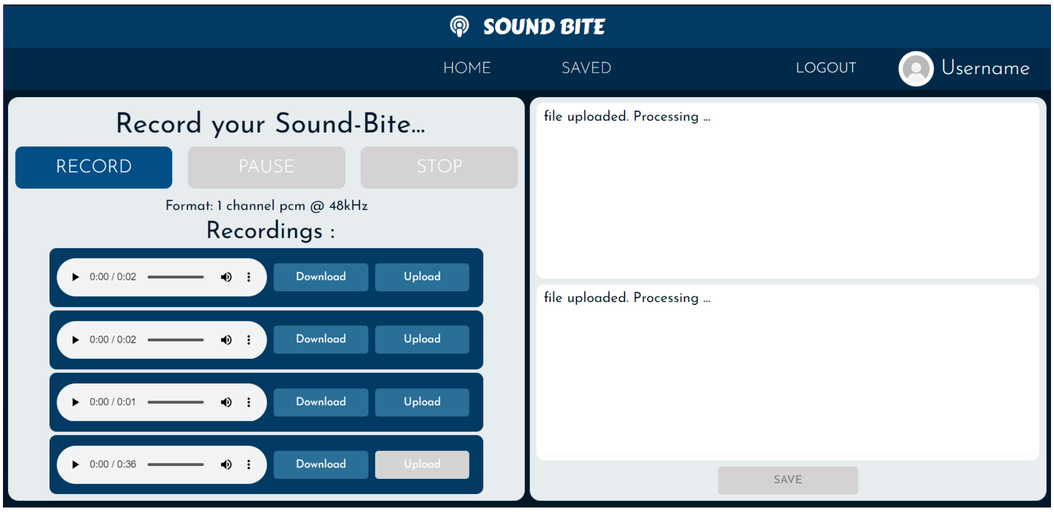

Figure 7.5: Home page - recordings

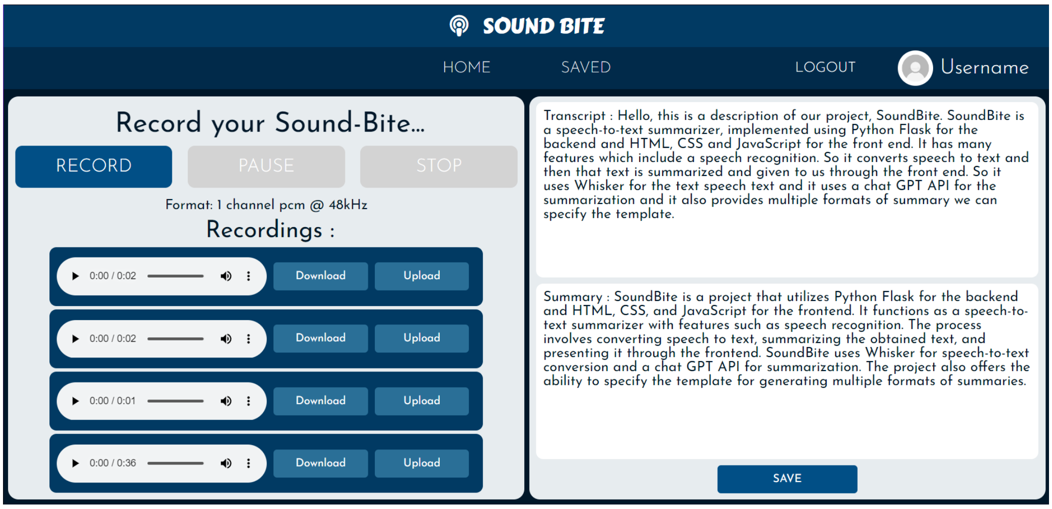

Figure 7.6: Home page - summary

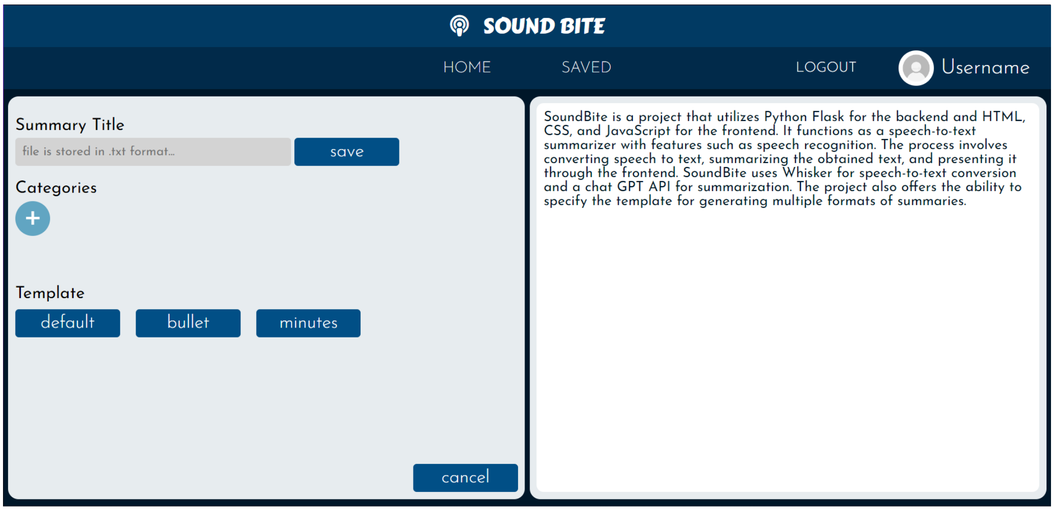

Figure 7.7: Save page

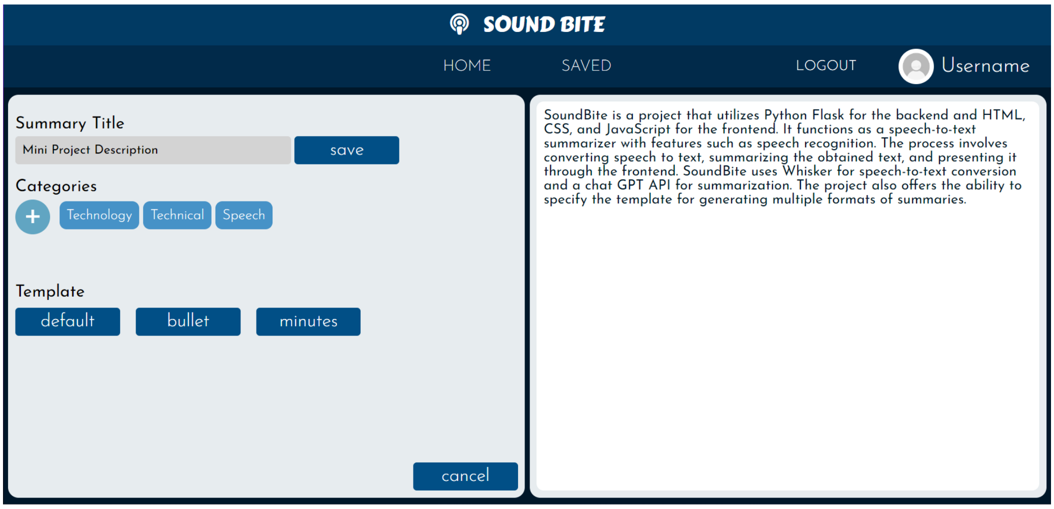

Figure 7.8: Save page - details

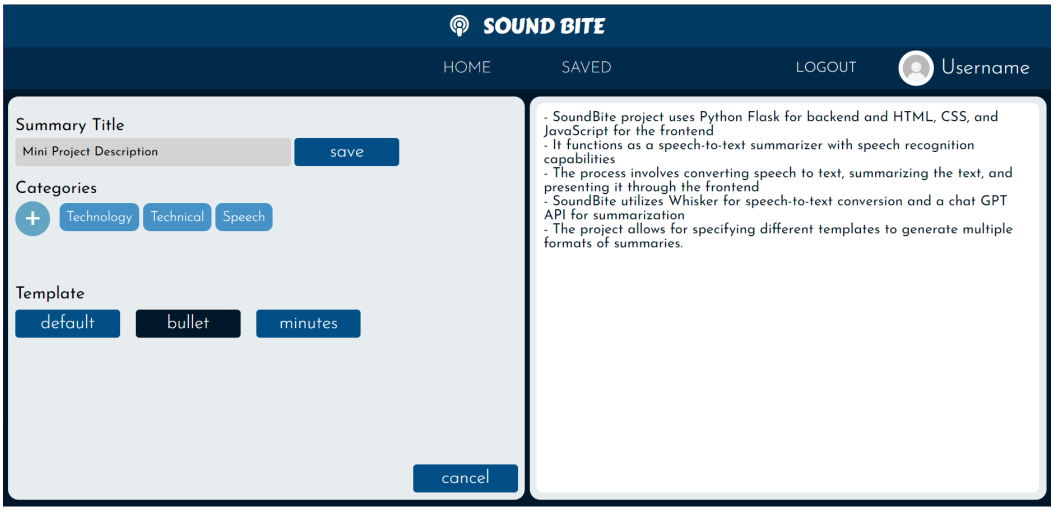

Figure 7.9: Save page - bullet template

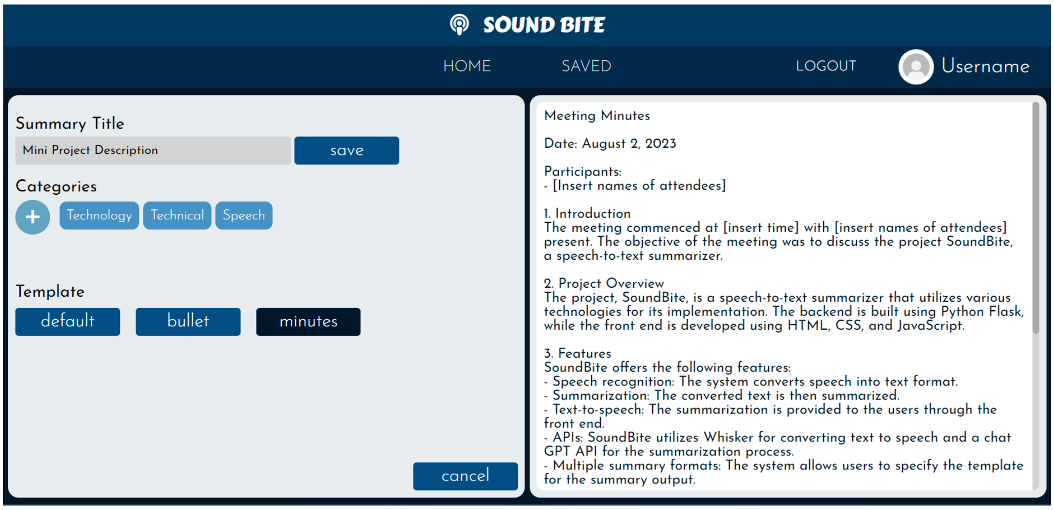

Figure 7.10: Save page - minutes template

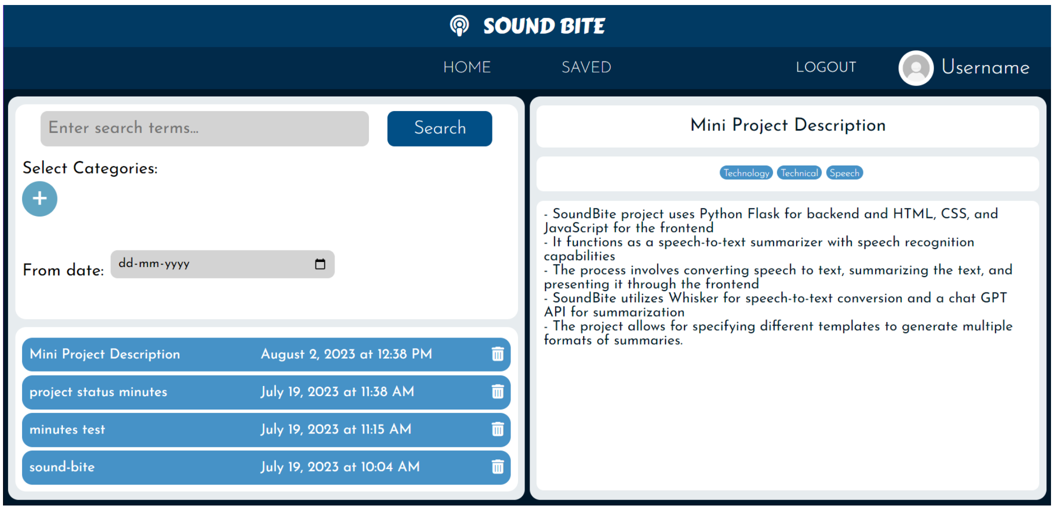

Figure 7.11: Saved page

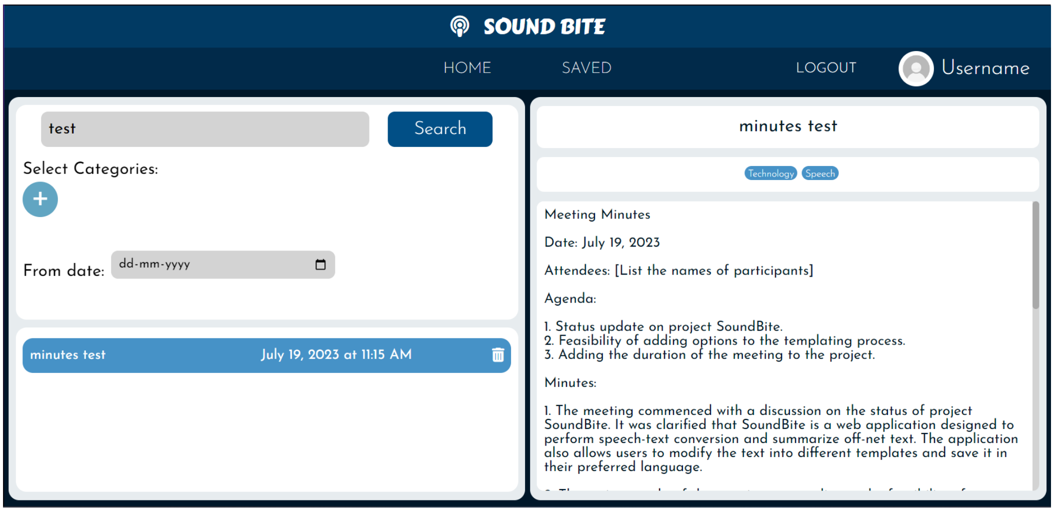

Figure 7.12: Saved page - search words

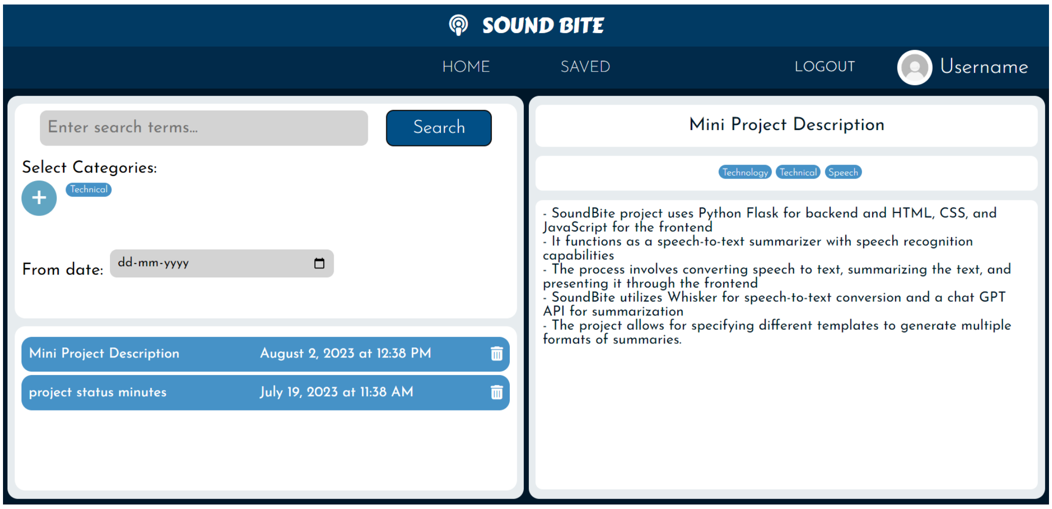

Figure 7.13: Saved page - search categories

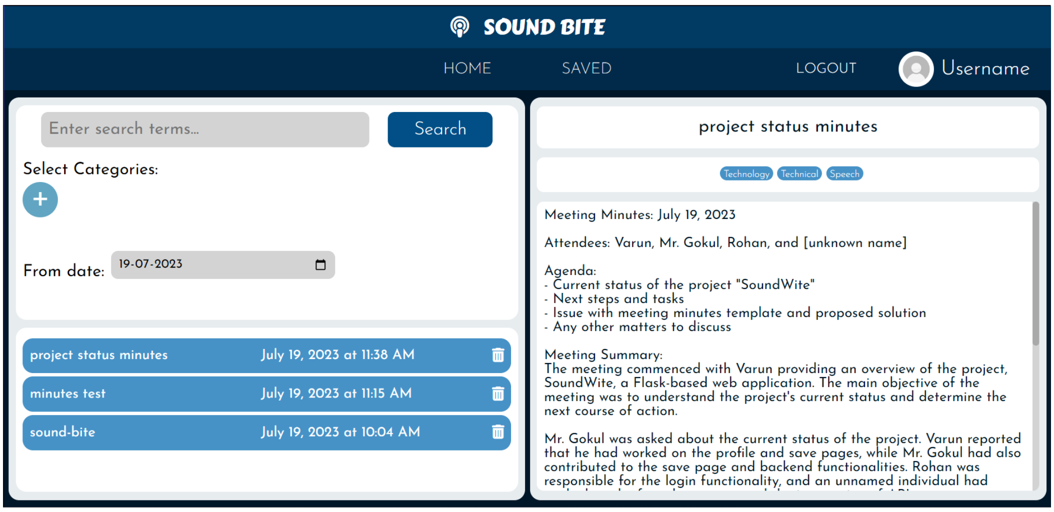

Figure 7.14: Saved page - search date

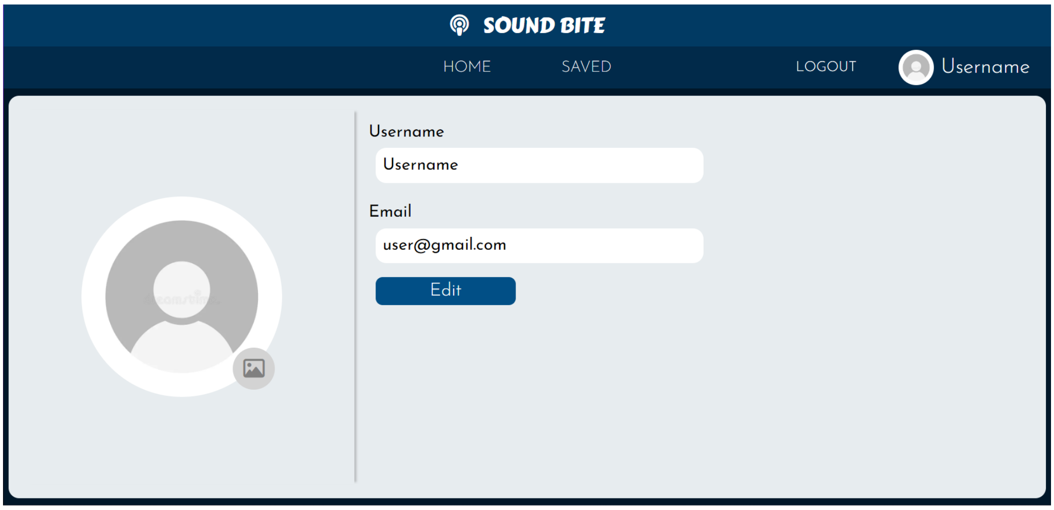

Figure 7.15: Profile page

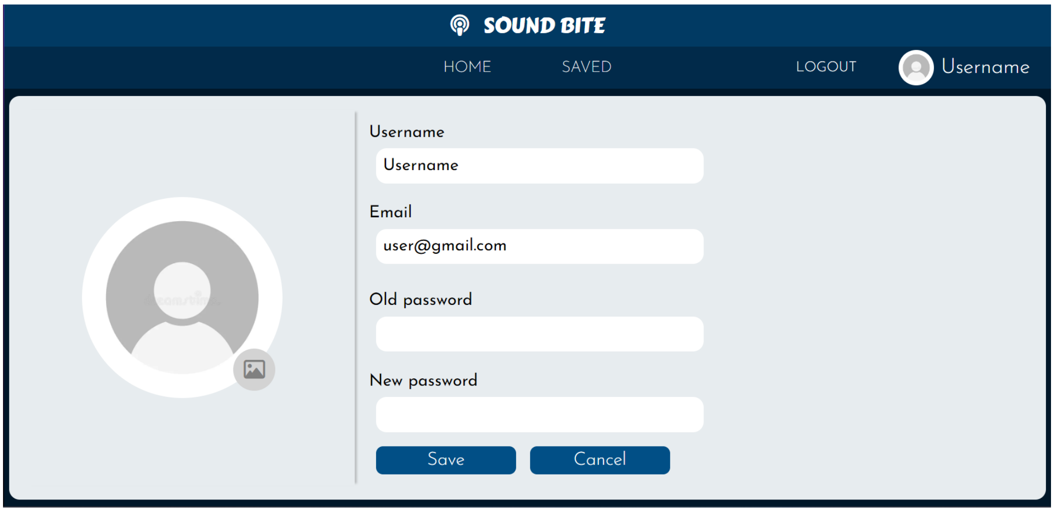

Figure 7.16: Profile page - edit

Chapter 8

Risks and Challenges

8.1 Key Risks and Challenges in Transcription:

1. Accent and Dialect Variations: Different speakers may have distinct accents and dialects, leading to variations in pronunciation and intonation.

2. Speech Rate: The system must accommodate speakers who speak at varying speeds, including fast speakers, and accurately capture their utterances.

3. Noisy Environments: Background noise and interference can hinder accurate transcription.

4. Homophones and Homographs: Ambiguous words with similar pronunciations but different meanings pose a challenge to the transcription process.

8.2 Key Risks and Challenges in Summarization:

1. Information Selection: The summarization algorithm must identify and extract the most relevant and salient information from the transcribed text.

2. Context Preservation: The generated summary should maintain the contextual coherence of the original content to ensure meaningful representation.

3. Length Constraint: The summary should be succinct while still conveying the essential aspects of the speech.

4. Co-reference Resolution: Resolving references to entities mentioned earlier in the text to avoid ambiguity in the summary.

Chapter 9

Conclusion

The development of an integrated Speech-to-Text and Text Summarization system presents a significant challenge that requires the exploration and application of advanced natural language processing (NLP) techniques, machine learning algorithms, and speech recognition technologies.

By addressing the challenges mentioned above and achieving high accuracy in both speech-to-text conversion and text summarization, the system can provide valuable assistance in various domains, including transcription services, content summarization, and knowledge extraction from audio resources.

References

[1] Building restful apis with flask in python. https://atmamani.github.io/blog/building-restful-apis-with-flask-in-python/.

[2] Openai gpt-3 api reference. https://platform.openai.com/docs/api-reference.

[3] Recorderjs github repository. https://github.com/mattdiamond/Recorderjs.

[4] Speechrecognition python package. https://pypi.org/project/SpeechRecognition/.

[5] Vinnarasu A and Deepa Jose. Speech to text conversion and summarization for effective understanding and documentation. *International Journal of Electrical and Computer Engineering (IJECE)*, 9:3642, 10 2019.

[6] John Conroy and Dianne O'leary. Text summarization via hidden markov models. pages 406–407, 09 2001.

[7] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, 2019.

[8] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision, 2022.

[9] Junjie Ye, Xuanting Chen, Nuo Xu, Can Zu, Zekai Shao, Shichun Liu, Yuhan Cui, Zeyang Zhou, Chao Gong, Yang Shen, Jie Zhou, Siming Chen, Tao Gui, Qi Zhang, and Xuanjing Huang. A comprehensive capability analysis of gpt-3 and gpt-3.5 series models, 2023.

[10] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization, 2020.

Appendix A: Sample Code

SAMPLE CODE

MAIN FLASK APP CODE

App.py

```
from flask import Flask, request, render_template, redirect, session, jsonify, url_for
from flask_pymongo import pymongo
import json
import requests
import time
import os
import speech_recognition as sr
from pydub import AudioSegment
import openai
from dotenv import load_dotenv
from flask_login import LoginManager, UserMixin, login_user, logout_user, current_user,
login_required
from datetime import datetime, timedelta
from bcrypt import hashpw, gensalt, checkpw

load_dotenv()


app = Flask(__name__)
app.secret_key = "very secret key"

# Flask-Login setup
login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = "login"

# MongoDB setup
CONNECTION_STRING =
"mongodb+srv://varunpradeep30:soundbite@sound-bite.uqvlkxi.mongodb.net/?retryWrites=true
&w=majority"
client = pymongo.MongoClient(CONNECTION_STRING)
db = client.get_database('Sound-bite')

openai.api_key = os.getenv("OPENAI_API_KEY")


class User(UserMixin):
    def __init__(self, email, username, password):
```

```
        self.email = email
        self.username = username
        self.password = password

    def get_id(self):
        return self.email


@login_manager.user_loader
def load_user(user_id):
    user_data = db.users.find_one({"email": user_id})
    if user_data:
        return User(user_data['email'], user_data['username'], user_data['password'])
    return None


def gptAPI(prompt, content):
    try:
        response = openai.ChatCompletion.create(
            model="gpt-3.5-turbo",
            messages=[
                {"role": "user", "content": "{} {}".format(prompt, content)}
            ]
        )
        data = response['choices'][0]['message']['content']
    except (Exception):
        data = "Server down"
    return data


@app.route("/")
def home():
    session["summary"] = ""
    session["transcript"] = ""
    if current_user.is_authenticated:
        return render_template("home.html", user=current_user.username)
    else:
        return render_template("home.html", user=None)


@app.route("/upload", methods=['POST'])
def upload():
    if request.method == "POST":
        f = request.files['audio_data']
```

```
        name = request.files['audio_data'].filename
        name = name.replace(":", "-")
        name = name.replace(".", "-") + '.wav'
        with open(name, 'wb') as audio:
            f.save(audio)
        print('file uploaded successfully')

        # transcription / summarization function calls here
        AUDIO_FILE = name
        # use the audio file as the audio source
        r = sr.Recognizer()
        with sr.AudioFile(AUDIO_FILE) as source:
            audio = r.record(source)  # read the entire audio file
            transcription = r.recognize_whisper(audio, language="english")

        # deleting audio file after use
        os.remove("./" + name)

        # summarizer
        prompt = "Please provide a summary of the following text (response should have no introductory phrases nor any chat like elements in the writing style of the summary. If no input text provided, respond exactly with no content to summarize). Here is the input text:"
        summary = gptAPI(prompt, transcription)

        session["summary"] = summary
        session["transcript"] = transcription

        data = {
            "transcript": transcription,
            "summary": summary
        }

        return data


@app.route("/template", methods=['POST'])
@login_required
def template():
    if request.method == "POST":
        data = request.json
        content = ""
        if data["template"] == "default":
            return session["summary"]
        if data["template"] == "bullet":
```

```
            content = session["summary"]
            prompt = "Insert the given summary into bullet points format (response should have no
introductory phrases nor any chat like elements in the writing style )"
        if data["template"] == "minutes":
            prompt = "Produce meeting minutes, given the date, and a transcript of the conversation
(in case the transcript is not in the format of a conversation, extract the main points for use in
meeting minutes) (follow standard meeting minutes format as closely as possible with the given
data). Here is the data : "
            content = "date: {} transcript: {}".format(data["date"], session["transcript"])

        #   prompt="Insert the given summary into minutes of a meeting format (response should
have no introductory phrases nor any chat like elements in the writing style )"
        summary = gptAPI(prompt, content)
        return summary


@app.route("/save", methods=['GET', 'POST'])
@login_required
def save():
    if session["summary"] == "":
        return redirect(url_for('home'))
    else:
        return render_template("save.html", summary=session["summary"],
user=current_user.username)


@app.route("/saveupload", methods=['GET', 'POST'])
def saveupload():
    if request.method == "POST":
        data = request.json
        db.summaries.insert_one(
            {"email": current_user.email, "title": data['title'], "categories": data['categories'],
"summary": data['summary'], "timestamp": data["timestamp"]})
        session["summary"] = ""
        return redirect(url_for('saved'))


@app.route("/login", methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('home'))

    if request.method == "POST":
        data = request.json
```

```
        user = load_user(data['email'])
        if user is None:
            return "email"
        elif not checkpw(data['password'].encode('utf-8'), user.password):
            return "password"
        else:
            login_user(user)
            return "Success"

    return render_template("login.html")


@app.route("/register", methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('home'))

    if request.method == "POST":
        data = request.json
        existing_user = load_user(data['email'])
        if existing_user is None:
            # Encrypt the password
            hashed_password = hashpw(
                data['password'].encode('utf-8'), gensalt())

            db.users.insert_one(
                {"email": data['email'], "username": data['username'], "password": hashed_password})
            return "Success"
        else:
            return "email"

    return render_template("register.html")


@app.route("/insert", methods=['GET', 'POST'])
def searchtest():
    if request.method == "POST":
        data = request.json
        db.summaries.insert_one({"email": data['email'], "title": data['title'],
                              "categories": data['categories'], "summary": data['summary'], "timestamp":
data["timestamp"]})
        return "success"
```

```
@app.route("/saved")
@login_required
def saved():
    return render_template("saved.html", user=current_user.username)


@app.route("/search", methods=['GET', 'POST'])
def search():
    if request.method == 'POST':
        data = request.json

        query = {}
        if data['input'] != "":
            query["$or"] = [
                {"title": {"$regex": data['input'], "$options": "i"}},
                {"summary": {"$regex": data['input'], "$options": "i"}}
            ]

        if data["categories"] != []:
            query["categories"] = {"$all": data["categories"]}
        if data['timestamp']:
            # Convert JavaScript timestamp to Python datetime
            timestamp_datetime = datetime.fromtimestamp(
                data["timestamp"] / 1000)
            # Divide by 1000 to convert milliseconds to seconds

            # Extract start and end dates
            start_date = timestamp_datetime.replace(
                hour=0, minute=0, second=0, microsecond=0)
            end_date = start_date + timedelta(days=1)
            query["timestamp"] = {"$gte": start_date.timestamp(
            ) * 1000, "$lt": end_date.timestamp() * 1000}

        query["email"] = current_user.email
        projection = {
            "title": 1,    # Include the 'title' field
            "summary": 1,  # Include the 'summary' field
            "categories": 1,
            "timestamp": 1,
            "_id": 0       # Exclude the '_id' field
        }

        response = db.summaries.find(query, projection)
        documents = list(response)
```

```
    return jsonify(documents)


@app.route("/profile")
@login_required
def profile():
    return render_template("profile.html", user=current_user.username)


@app.route("/logout")
@login_required
def logout():
    logout_user()
    session.pop('summary')
    return redirect(url_for('home'))


if __name__ == '__main__':
    app.run(debug=True, port=8000)
```

Appendix B: CO-PO And CO-PSO Mapping

COURSE OUTCOMES:

After completion of the course the student will be able to

SL. NO	**DESCRIPTION**	**Blooms' Taxonomy Level**
CO1	Identify technically and economically feasible problems (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO2	Identify and survey the relevant literature for getting exposed to related solutions and get familiarized with software development processes (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO3	Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions of minimal complexity by using modern tools & advanced programming techniques (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO4	Prepare technical report and deliver presentation (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO5	Apply engineering and management principles to achieve the goal of the project (Cognitive Knowledge Level: Apply)	Level 3: Apply

CO-PO AND CO-PSO MAPPING

	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO3
CO1	3	3	3	3		2	2	3	2	2	2	3	2	2	2
CO2	3	3	3	3	3	2		3	2	3	2	3	2	2	2
CO3	3	3	3	3	3	2	2	3	2	2	2	3			2
CO4	2	3	2	2	2			3	3	3	2	3	2	2	2
CO5	3	3	3	2	2	2	2	3	2		2	3	2	2	2

3/2/1: high/medium/low

JUSTIFICATIONS FOR CO-PO MAPPING

MAPPING	LOW/ MEDIUM/ HIGH	JUSTIFICATION
100003/CS622T.1-PO1	**HIGH**	Identify technically and economically feasible problems by applying the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS622T.1-PO2	**HIGH**	Identify technically and economically feasible problems by analysing complex engineering problems reaching substantiated conclusions using first principles of mathematics.
100003/CS622T.1-PO3	**HIGH**	Design solutions for complex engineering problems by identifying technically and economically feasible problems.
100003/CS622T.1-PO4	**HIGH**	Identify technically and economically feasible problems by analysis and interpretation of data.
100003/CS622T.1-PO6	**MEDIUM**	Responsibilities relevant to the professional engineering practice by identifying the problem.
100003/CS622T.1-PO7	**MEDIUM**	Identify technically and economically feasible problems by understanding the impact of the professional engineering solutions.
100003/CS622T.1-PO8	**HIGH**	Apply ethical principles and commit to professional ethics to identify technically and economically feasible problems.
100003/CS622T.1-PO9	**MEDIUM**	Identify technically and economically feasible problems by working as a team.
100003/CS622T.1-PO10	**MEDIUM**	Communicate effectively with the engineering community by identifying technically and economically feasible problems.
100003/CS622T.1-P011	**MEDIUM**	Demonstrate knowledge and understanding of engineering and management principles by selecting the technically and economically feasible problems.
100003/CS622T.1-PO12	**HIGH**	Identify technically and economically feasible problems for long term learning.
100003/CS622T.1-PSO1	**MEDIUM**	Ability to identify, analyze and design solutions to identify technically and economically feasible problems.
100003/CS622T.1-PSO2	**MEDIUM**	By designing algorithms and applying standard practices in software project development and Identifying technically and economically feasible problems.
100003/CS622T.1-PSO3	**MEDIUM**	Fundamentals of computer science in competitive research can be applied to Identify technically and economically feasible problems.
100003/CS622T.2-PO1	**HIGH**	Identify and survey the relevant by applying the knowledge of mathematics, science, engineering fundamentals.

100003/CS622T.2-PO2	**HIGH**	Identify, formulate, review research literature, and analyze complex engineering problems get familiarized with software development processes.
100003/CS622T.2-PO3	**HIGH**	Design solutions for complex engineering problems and design based on the relevant literature.
100003/CS622T.2-PO4	**HIGH**	Use research-based knowledge including design of experiments based on relevant literature.
100003/CS622T.2-PO5	**HIGH**	Identify and survey the relevant literature for getting exposed to related solutions and get familiarized with software development processes by using modern tools.
100003/CS622T.2-PO6	**MEDIUM**	Create, select, and apply appropriate techniques, resources, by identifying and surveying the relevant literature.
100003/CS622T.2-PO8	**HIGH**	Apply ethical principles and commit to professional ethics based on the relevant literature.
100003/CS622T.2-PO9	**MEDIUM**	Identify and survey the relevant literature as a team.
100003/CS622T.2-PO10	**HIGH**	Identify and survey the relevant literature for a good communication to the engineering fraternity.
100003/CS622T.2-PO11	**MEDIUM**	Identify and survey the relevant literature to demonstrate knowledge and understanding of engineering and management principles.
100003/CS622T.2-PO12	**HIGH**	Identify and survey the relevant literature for independent and lifelong learning.
100003/CS622T.2-PSO1	**MEDIUM**	Design solutions for complex engineering problems by Identifying and survey the relevant literature.
100003/CS622T.2-PSO2	**MEDIUM**	Identify and survey the relevant literature for acquiring programming efficiency by designing algorithms and applying standard practices.
100003/CS622T.2-PSO3	**MEDIUM**	Identify and survey the relevant literature to apply the fundamentals of computer science in competitive research.
100003/CS622T.3-PO1	**HIGH**	Perform requirement analysis, identify design methodologies by using modern tools & advanced programming techniques and by applying the knowledge of mathematics, science, engineering fundamentals.
100003/CS622T.3-PO2	**HIGH**	Identify, formulate, review research literature for requirement analysis, identify design methodologies and develop adaptable & reusable solutions.

100003/CS622T.3-PO3	**HIGH**	Design solutions for complex engineering problems and perform requirement analysis, identify design methodologies.
100003/CS622T.3-PO4	**HIGH**	Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
100003/CS622T.3-PO5	**HIGH**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools.
100003/CS622T.3-PO6	**MEDIUM**	Perform requirement analysis, identify design methodologies and assess societal, health, safety, legal, and cultural issues.
100003/CS622T.3-PO7	**MEDIUM**	Understand the impact of the professional engineering solutions in societal and environmental contexts and Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions.
100003/CS622T.3-PO8	**HIGH**	Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions by applying ethical principles and commit to professional ethics.
100003/CS622T.3-PO9	**MEDIUM**	Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.
100003/CS622T.3-PO10	**MEDIUM**	Communicate effectively with the engineering community and with society at large to perform requirement analysis, identify design methodologies.
100003/CS622T.3-PO11	**MEDIUM**	Demonstrate knowledge and understanding of engineering requirement analysis by identifying design methodologies.
100003/CS622T.3-PO12	**HIGH**	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change by analysis, identify design methodologies and develop adaptable & reusable solutions.
100003/CS622T.3-PSO3	**MEDIUM**	The ability to apply the fundamentals of computer science in competitive research and prior to that perform requirement analysis, identify design methodologies.
100003/CS622T.4-PO1	**MEDIUM**	Prepare technical report and deliver presentation by applying the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS622T.4-PO2	**HIGH**	Identify, formulate, review research literature, and analyze complex engineering problems by preparing technical report and deliver presentation.

100003/CS622T.4-PO3	**MEDIUM**	Prepare Design solutions for complex engineering problems and create technical report and deliver presentation.
100003/CS622T.4-PO4	**MEDIUM**	Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions and prepare technical report and deliver presentation.
100003/CS622T.4-PO5	**MEDIUM**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools and Prepare technical report and deliver presentation.
100003/CS622T.4-PO8	**HIGH**	Prepare technical report and deliver presentation by applying ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
100003/CS622T.4-PO9	**HIGH**	Prepare technical report and deliver presentation effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.
100003/CS622T.4-PO10	**HIGH**	Communicate effectively with the engineering community and with society at large by prepare technical report and deliver presentation.
100003/CS622T.4-PO11	**MEDIUM**	Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work by prepare technical report and deliver presentation.
100003/CS622T.4-PO12	**HIGH**	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change by prepare technical report and deliver presentation.
100003/CS622T.4-PSO1	**MEDIUM**	Prepare a technical report and deliver presentation to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas.
100003/CS622T.4-PSO2	**MEDIUM**	To acquire programming efficiency by designing algorithms and applying standard practices in software project development and to prepare technical report and deliver presentation.
100003/CS622T.4-PSO3	**MEDIUM**	To apply the fundamentals of computer science in competitive research and to develop innovative products to meet the societal needs by preparing technical report and deliver presentation.
100003/CS622T.5-PO1	**HIGH**	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS622T.5-PO2	**HIGH**	Identify, formulate, review research literature, and analyze complex engineering problems by applying engineering and management principles to achieve the goal of the project.

100003/CS622T.5-PO3	**HIGH**	Apply engineering and management principles to achieve the goal of the project and to design solutions for complex engineering problems and design system components or processes that meet the specified needs.
100003/CS622T.5-PO4	**MEDIUM**	Apply engineering and management principles to achieve the goal of the project and use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
100003/CS622T.5-PO5	**MEDIUM**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO6	**MEDIUM**	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities by applying engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO7	**MEDIUM**	Understand the impact of the professional engineering solutions in societal and environmental contexts, and apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO8	**HIGH**	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice and to use the engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO9	**MEDIUM**	Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO11	**MEDIUM**	Demonstrate knowledge and understanding of engineering and management principles and apply these to one’s own work, as a member and leader in a team. Manage projects in multidisciplinary environments and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO12	**HIGH**	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PSO1	**MEDIUM**	The ability to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas. Apply engineering and management principles to achieve the goal of the project.

100003/CS622T.5-PSO2	**MEDIUM**	The ability to acquire programming efficiency by designing algorithms and applying standard practices in software project development to deliver quality software products meeting the demands of the industry and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PSO3	**MEDIUM**	The ability to apply the fundamentals of computer science in competitive research and to develop innovative products to meet the societal needs thereby evolving as an eminent researcher and entrepreneur and apply engineering and management principles to achieve the goal of the project.