Project Report : Voc-Notes

INFO 607 Spring 2024

**Members:** Imon Bera, Precious Orekha, Priyanka Patil, Vijval Vemula

|  |  |  |
| --- | --- | --- |
| Full Name | Email ID | Signature |
| Imon Bera | ib385@drexel.edu | IB |
| Precious Orekha | poo27@drexel.edu | PO |
| Priyanka Patil | pp673@drexel.edu | PP |
| Vijval Vemula | vv354@drexel.edu | VV |

# Project Overview

## Introduction

Voc-Notes is a project designed to capture, transcribe, and process audio recordings of lectures into structured, insightful educational content. The application leverages advanced AI models, Google Cloud Speech-to-Text, and data visualization tools to provide users with concise and useful class notes.

## Purpose

The primary purpose of Voc-Notes is to help students by providing an automated solution for capturing lecture content and converting it into a readable and structured format. This allows students to focus on understanding the lecture instead of taking notes.

## Key Features

Key features of the Voc-Notes project include:

1. **Audio Capture/Upload:** The first component is the capture or upload of live lecture audio. This can be done in real-time during lectures or by uploading pre-recorded audio files.
2. **Audio-to-Text Conversion:** Once the audio is captured or uploaded, it is converted into text using Google Cloud Speech-to-Text. This powerful tool ensures accurate transcription of spoken words into text form.
3. **Data Preprocessing:** After converting the audio to text, the data needs to be cleaned and preprocessed. This is achieved using PySpark, which allows for efficient handling of large volumes of text data, ensuring it is in a suitable format for further processing.
4. **Summarization:** The core of our system is the summarization process. Here, we use groq inference on mixtral and llama 3 models to summarize the text into concise class notes. This step is crucial for distilling the key information from the lectures.
5. **Data Storage:** The processed data is then stored in MongoDB. This NoSQL database is ideal for handling the varied and voluminous data generated from lecture recordings and their transcriptions.
6. **Data Visualization:** Finally, we use Tableau hyper API to extract data for visualization. This component helps in visualizing the summarized information, providing valuable insights and making it easier for users to understand and utilize the educational content.

Each of these components plays a vital role in ensuring our system is efficient, scalable, and capable of providing high-quality educational content from live lecture audio.

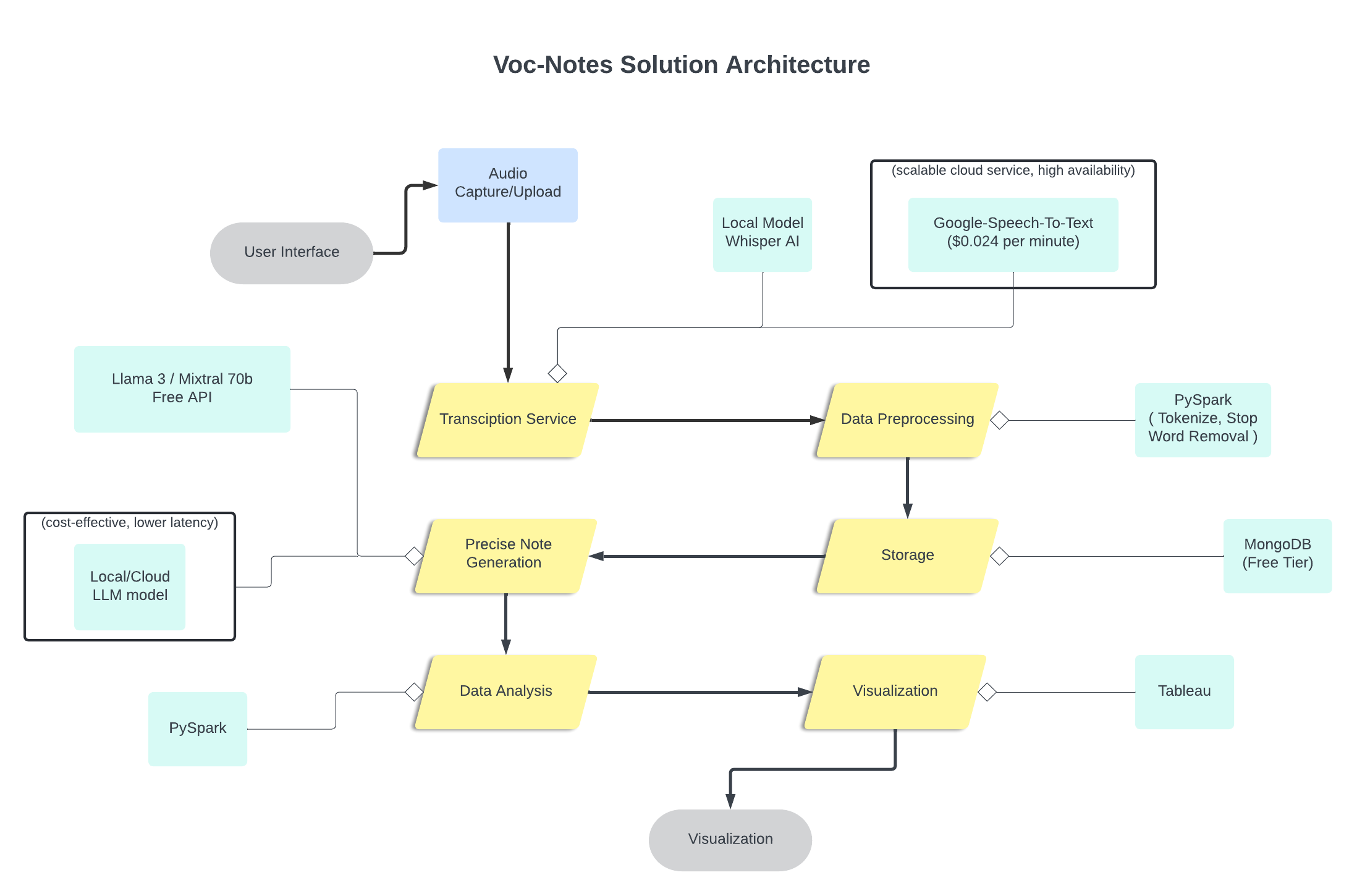
* **Technologies Used for this Project**

1. **Streamlit for UI (front-end):** We chose Streamlit to create our user interface because it allows for rapid development and easy deployment of interactive web applications. This helps users interact with our system seamlessly.
2. **Whisper AI:** Whisper AI is employed for initial processing of the audio, ensuring high-quality audio capture and noise reduction, which is crucial for accurate transcription.
3. **Google Cloud Speech-to-Text:** This service is used to convert audio into text. Google Cloud Speech-to-Text is known for its accuracy and efficiency in transcribing spoken language into written form.
4. **PySpark:** For data preprocessing, we use PySpark. It handles large datasets efficiently, cleaning and preparing the text data for the summarization step.
5. **Ollama:** This tool aids in managing and orchestrating the large language models (LLMs) we utilize, ensuring smooth and efficient operations during text summarization.
6. **LLMs: Mixtral-8x7b (32768 token size) & Llama3-70b (8192 token size):** We leverage these advanced large language models to perform the summarization of the text. Mixtral-8x7b and Llama3-70b are chosen for their high token capacity and ability to generate concise, meaningful summaries.
7. **PyMongo:** For data storage, we use PyMongo to interact with MongoDB. This allows us to store processed data efficiently and retrieve it as needed for further analysis or visualization.
8. **Tableau:** Finally, we use Tableau for data visualization. This tool helps us create insightful and interactive visualizations, making it easier for users to understand and engage with the summarized educational content.

By integrating these technologies, we tried to make our project robust, efficient, and capable of delivering high-quality educational content from live lecture audio.

**System Architecture**  
The Voc-Notes system architecture comprises several components that work together to capture, process, and visualize lecture audio data.   
The following sections provide detailed descriptions of each component in the flow.

## **Architecture Diagram** The architecture diagram illustrates the workflow of the Voc-Notes system, highlighting the interaction between different components.

  
**User Interface:**

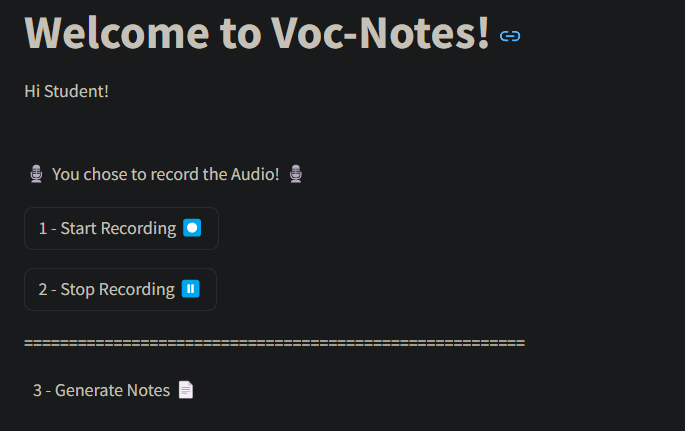
Our application frontend serves as the entry point for users. It greets the user and provides intuitive links to the core functionalities of the system: the Record Audio page and the Upload Audio page.

* **Greets the User:** When users access the application, they are greeted with a welcoming interface.
* **Navigation Links:** The main page provides clear links to the Record Audio and Upload Audio pages, ensuring users can easily find and access the features they need."

**Record Audio Page:**

We have the Record Audio page, which offers users the ability to record their audio directly within the application.

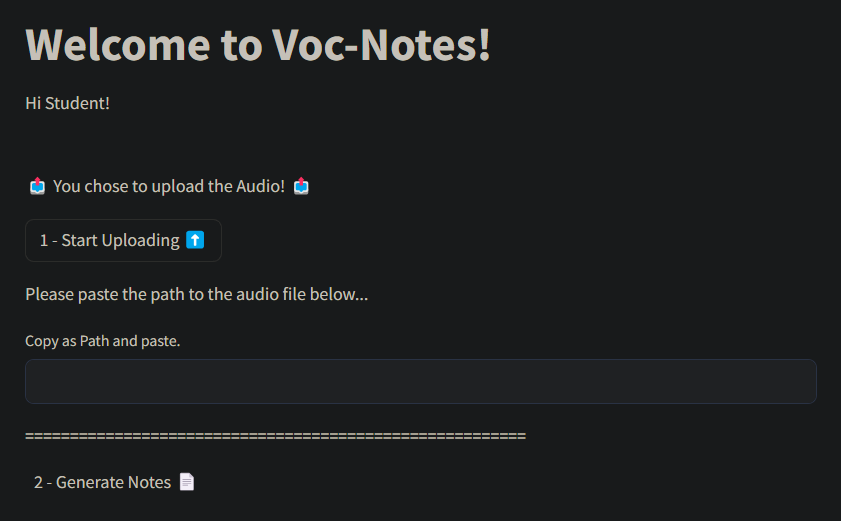
* **Start and Stop Recording:** The page displays simple options to start and stop audio recording, making it easy for users to capture live lecture audio.
* **Link to Generate Notes:** Once the recording is complete, users can easily navigate to the Generate Notes page via a provided link, streamlining the workflow from recording to note generation.



**Upload Audio Page:**

The Upload Audio page caters to users who already have audio files they wish to process.

* **Local Path Input:** The page prompts users to enter the local path of their audio file, making the upload process straightforward.
* **Generate Notes Button:** After specifying the file path, users can initiate the note generation process with a simple button click. This starts the processing of the uploaded audio, seamlessly integrating with our backend services.



**Audio Capture/Upload:**

The system provides flexible options for audio capture, allowing users to either upload an existing audio file or record new audio directly within the application.

* **Uploading an Audio File:** Users can specify the path to an existing audio file, which is subsequently processed by the system.
* **Recording New Audio:** The system also enables users to record new audio, offering a seamless and integrated user experience.

**Recording Audio**

The process of recording audio in the application is designed to be straightforward and efficient.

* **Capturing Audio:** The system captures audio through an input stream, enabling real-time audio input from the user’s device.
* **Storing in a Queue:** During the capture process, the audio data is stored in a queue, ensuring continuous recording without data loss.

**Start and Stop Recording**

The recording process is managed by start and stop functions, providing users with clear and easy-to-use controls.

* **Start Recording:** When the user initiates the recording by selecting the 'Start Recording' option, the system begins capturing audio immediately and storing it in the queue.
* **Stop Recording:** Upon selecting the 'Stop Recording' option, the audio capture ceases. The system then normalizes the audio, converts it to the appropriate format, and saves it as an MP3 file named recorded\_audio.mp3.

These functionalities ensure that users can easily capture and manage their audio recordings. The system efficiently handles both uploaded files and newly recorded audio, preparing the audio for subsequent processing and note generation.

**Transcription Service/ Speech-to-Text Conversion**

To facilitate the conversion of speech to text, the system initializes the Google Speech-to-Text client with the necessary API key.

**Audio Processing**

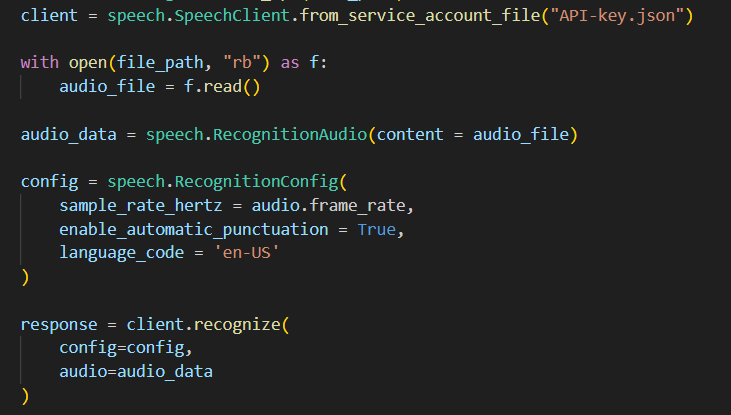
The audio processing stage involves several critical steps to ensure the audio is ready for transcription:

* **Chunk Conversion:** The MP3 audio file is segmented into smaller chunks to comply with the API's size limitations.
* **Data Preparation:** The system reads and prepares the audio data for recognition. This preparation includes setting the sample rate, enabling automatic punctuation, and specifying the appropriate language code.

**Transcription**

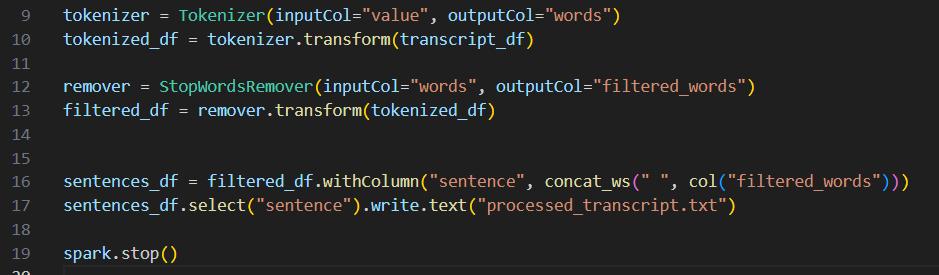
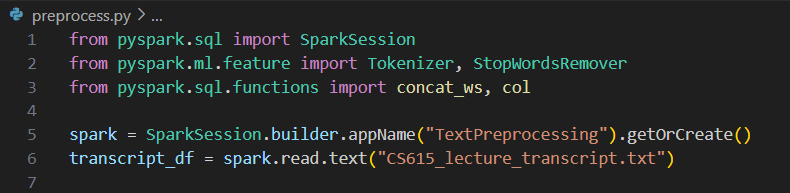
The transcription stage encompasses the following:

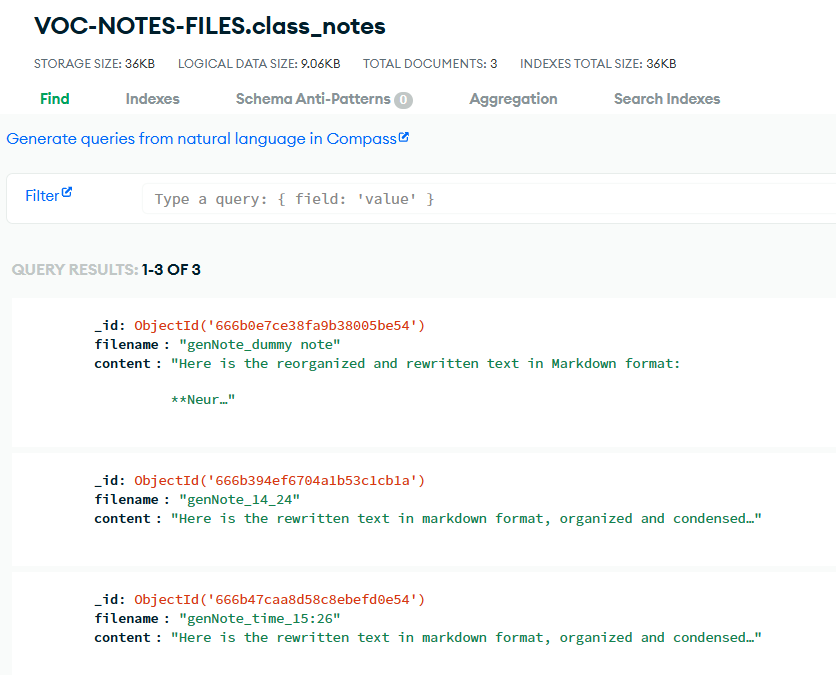
* **API Submission:** The prepared audio chunks are submitted to the Google Speech-to-Text API.
* **Text Retrieval:** The API processes the audio and returns the corresponding transcribed text.



By following this structured process, the system ensures an accurate and efficient transformation of audio content into text format, leveraging the robust capabilities of the Google Speech-to-Text service.

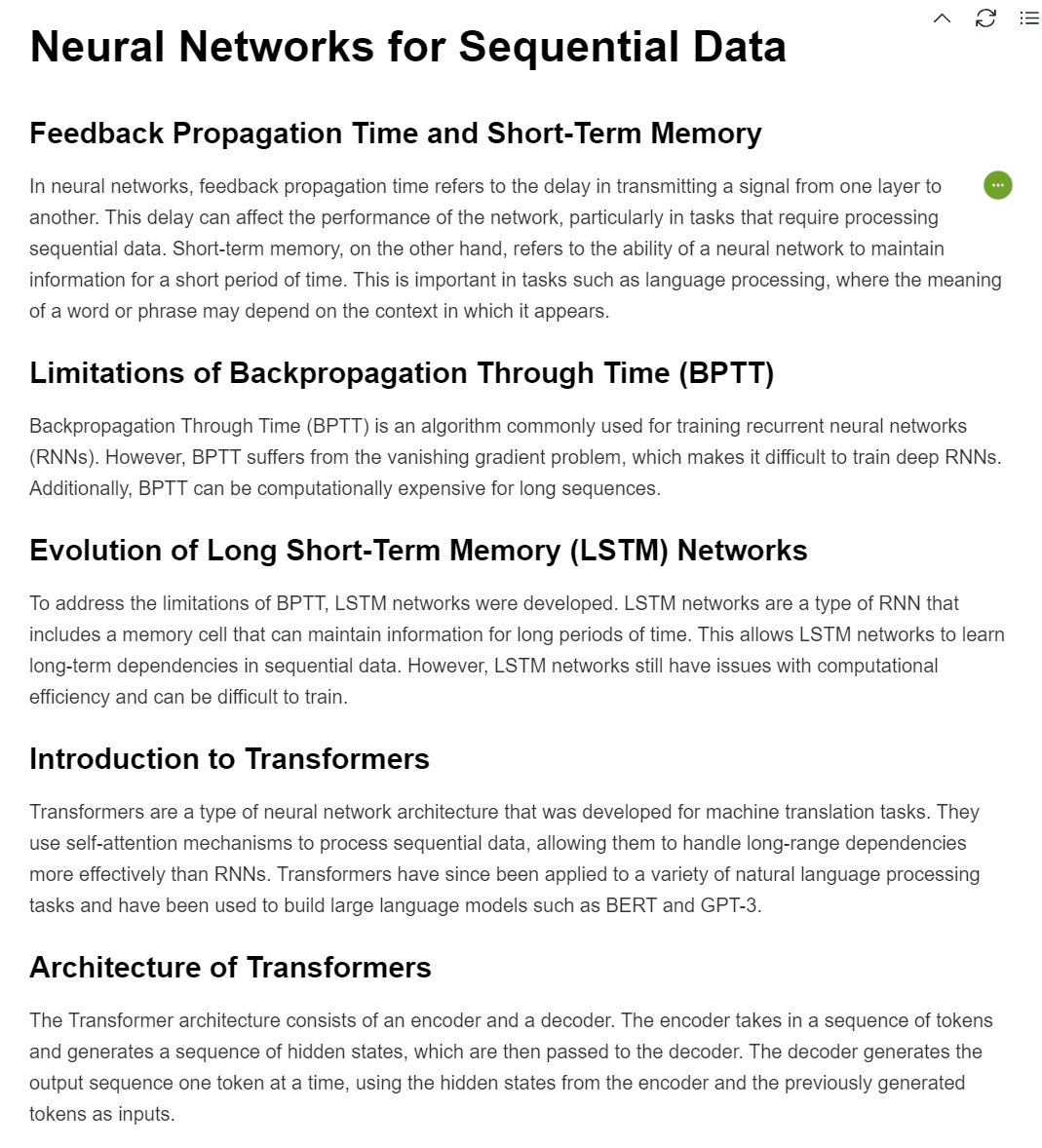
**Data Preprocessing:**  
Data preprocessing is a vital step in preparing the transcribed text for subsequent analysis and summarization. This component employs PySpark to handle large volumes of text data efficiently. The preprocessing steps include tokenization, which breaks the text into individual words or tokens, and stop word removal, which filters out common words that do not contribute significant meaning, such as "and," "the," and "is." By cleaning and organizing the text data, the preprocessing component ensures that the text is in an optimal state for accurate and meaningful summarization. The processed text is then ready for the note generation phase.



**Storage:**  
The storage component is responsible for saving and managing the processed data within the Voc-Notes system. This component utilizes MongoDB, a NoSQL database, to store transcribed text, processed data, and generated notes. MongoDB was chosen for its scalability, flexibility, and ability to handle unstructured data efficiently. Each entry in the database is timestamped, ensuring that data is organized chronologically and can be retrieved easily. The storage component also includes mechanisms for efficient querying and retrieval, enabling users to access their notes and transcripts as needed. This reliable data management is essential for maintaining the integrity and accessibility of information. Here is image of the database with few class notes populated,  
  


**Precise Note Generation:**  
The precise note generation component is where the core value of the Voc-Notes system is realized. This component uses advanced language models such as Llama 3 and Mixtral 70b to summarize the preprocessed text into concise, coherent class notes. The summarization process involves splitting the text into manageable chunks, generating summaries for each chunk, and then combining these summaries into a final, polished document. The use of powerful language models ensures that the generated notes are not only accurate but also capture the essential points and nuances of the lecture. This component significantly reduces the time and effort required for students to create comprehensive study notes.

Here is a sample of the generated note from a Deep Learning Lecture -



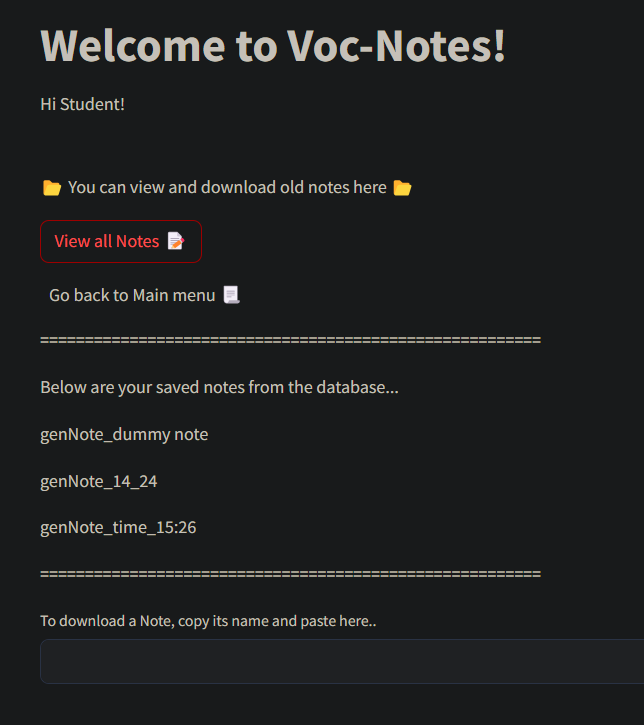
**Data Storage:**

MongoDB is used to store both the processed lecture transcripts and the generated class notes. The database collections are organized to efficiently manage and retrieve notes.

Benefits:

* Scalability: MongoDB allows the application to handle increasing amounts of data seamlessly as more notes are added.
* Efficient Data Retrieval: The use of MongoDB's indexing and querying capabilities ensures fast access to stored notes, enhancing user experience.

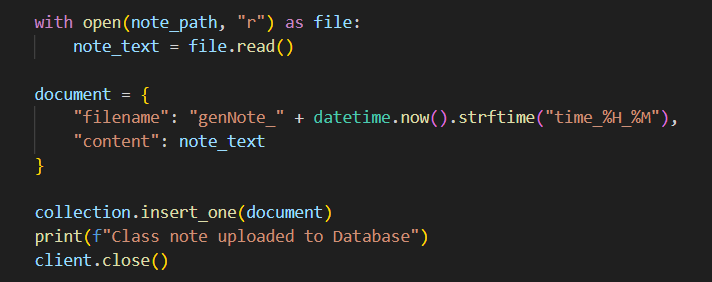
Since we developed the UI with Streamlit, it provides an intuitive and interactive web interface for students to view and download their notes. The interface includes buttons and text inputs for easy navigation and interaction.



Users can click a button to view all saved notes retrieved from MongoDB. Also, they can enter the name of a note to download it directly from the database.

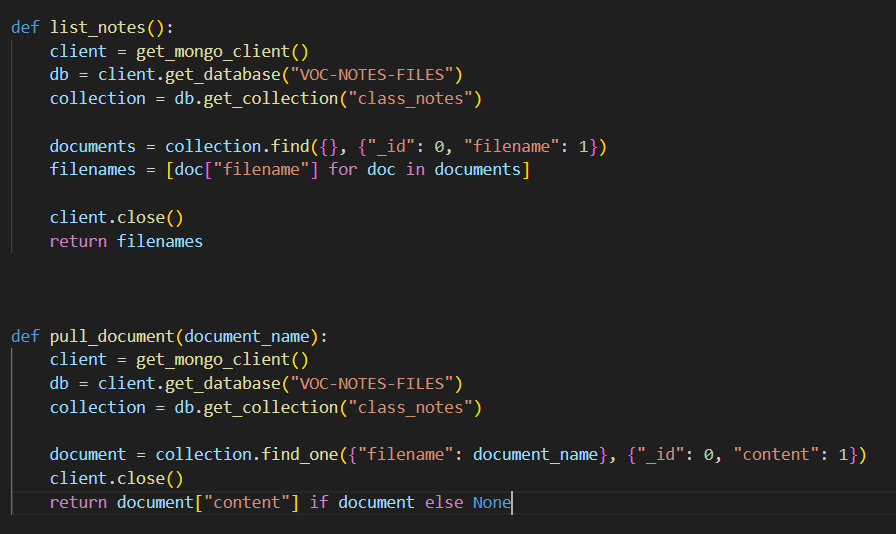
**Automatic Filename Generation:**

Uploaded notes and transcripts are stored with filenames that include timestamps, ensuring unique and organized storage.



**List and Retrieve Notes:**

* The `list\_notes` function fetches all note filenames from the MongoDB collection for display.
* The `pull\_document` function retrieves the content of a specific note based on its filename.

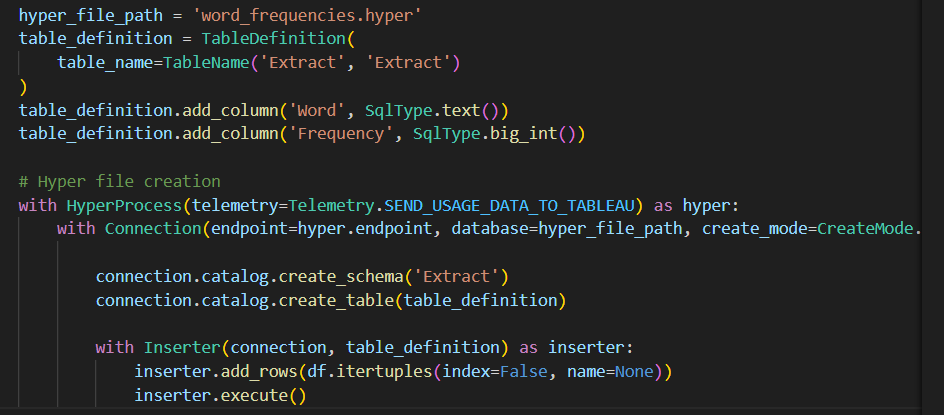


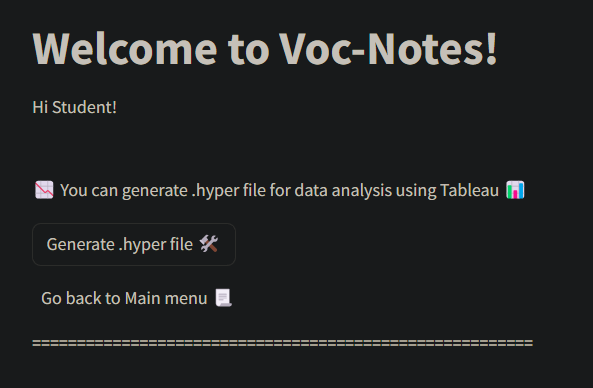
**Data Analysis:**

Python scripts are used to preprocess the lecture transcripts, tokenize text, remove stop words, and calculate word frequencies. The data is then stored in a `.hyper` file format for efficient analysis in Tableau. The benefits would be,

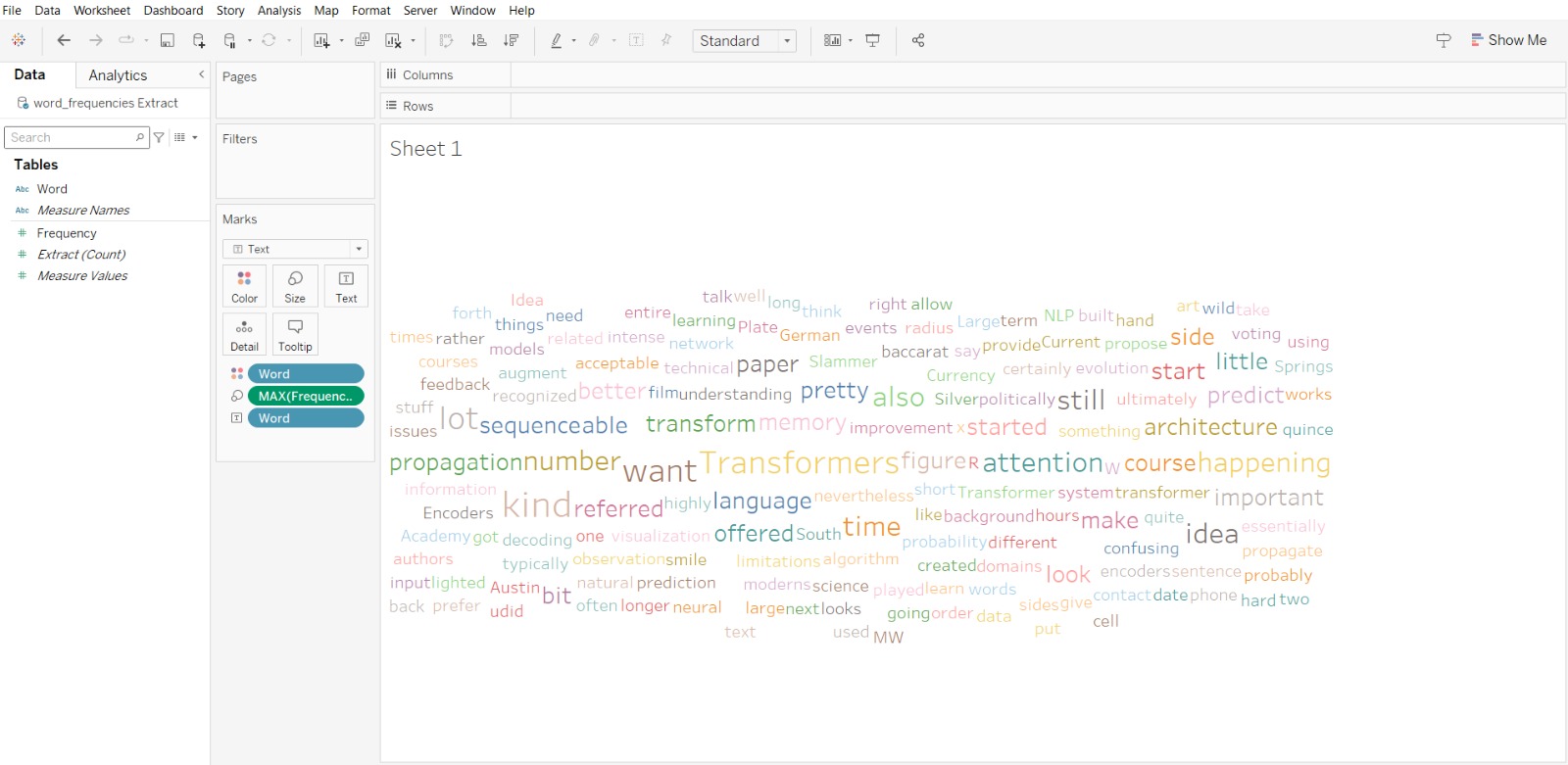
* **Automated Text Processing:** Using Python libraries like `nltk` for tokenization and stop word removal, and `Counter` for word frequency calculation.
* **Integration with Tableau:** The processed data is exported to a `.hyper` file, which can be easily imported into Tableau for advanced data visualization and analysis.

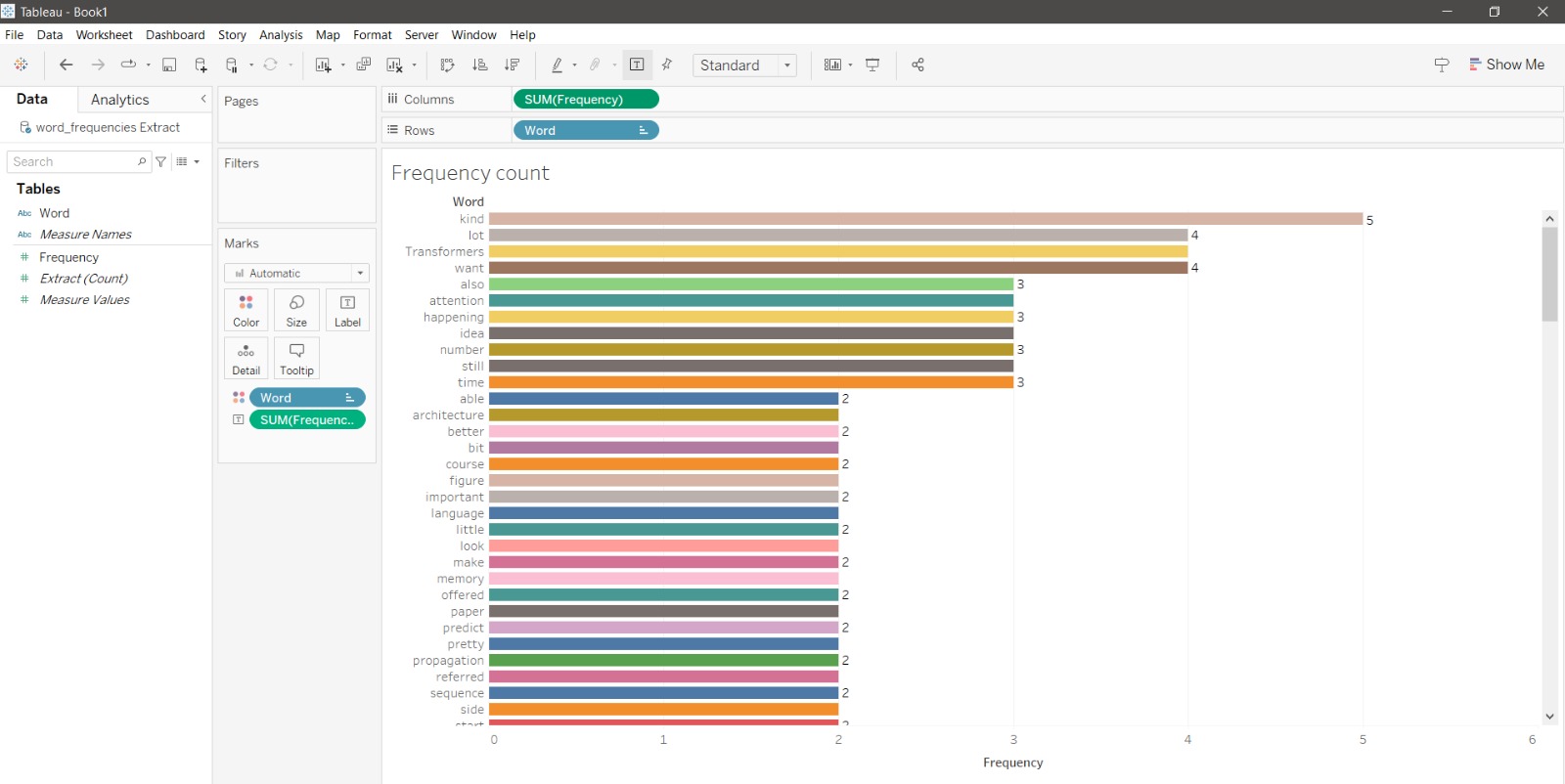
Creating a `.hyper` File using Tableau Hyper API:



This data file generation is integrated into the software. After each note generation the user can request a .hyper file by going into the 4th option.  
  


After generating the hyper file, users can import that seamlessly to their copy of Tableau and use various data analysis to pull insights from the lecture.

Here below we created a word cloud from the hyper file of a Deep Learning Lecture using Tableau,  
  


And a word frequency table to get a rough idea on the hot topic of the lecture.  
  


**System Workflow Decision:**

**Transcription Service Decision:**

In deciding the best transcription service for our application, we considered two main options: Whisper AI as a local model and Google Speech-to-Text. Let's discuss about the pros and cons of each option and the rationale behind our final decision.

**Whisper AI (Local Model):**

* **Pros:** Whisper AI is cost-efficient as it incurs no additional service charges. This makes it an attractive option for projects with budget constraints.
* **Cons:** However, it is resource-heavy for local systems. Running Whisper AI locally requires significant computational power and memory, which can be a limiting factor, especially for real-time processing of large volumes of audio data.

**Google Speech-to-Text:**

* **Pros:** On the other hand, Google Speech-to-Text offers high availability and scalability. It is a cloud-based service, which means it can handle large and variable workloads efficiently. Additionally, it is cost-effective, priced at $0.024 per minute of audio, making it a feasible option for continuous and large-scale use.
* **Cons:** The primary downside is the service charges that apply, which can accumulate over time.

**Decision:** After weighing the pros and cons, we chose Google Speech-to-Text for our system. The key factors influencing our decision were its scalability and the reduced resource demand on local systems. By leveraging a cloud-based solution, we ensure our system can handle high volumes of audio data without the need for extensive local computational resources. This choice aligns with our goals of efficiency and scalability, ensuring that our system remains robust and responsive as it grows.

**LLM Model API Decision:**

For our LLM model API, we considered Groq API and Ollama API (local model).

**Groq API:**

* **Pros:** The Groq API is an external API, which means it does not burden local resources. This allows us to offload the heavy computational work to external servers, ensuring that our local systems remain efficient and responsive.
* **Cons:** However, this option makes us dependent on the availability and reliability of the external service. Additionally, there could be associated costs depending on usage levels, which need to be managed.

**Ollama API (Local Model):**

* **Pros:** Using the Ollama API as a local model provides us with complete control over the model and no external dependencies. This ensures that we can operate independently of external service providers.
* **Cons:** The major drawback is that it is resource-heavy for local systems. Running such a large model locally requires significant computational power and memory, which can be challenging, especially for real-time applications.
* **Decision:** After considering these factors, we decided to use the Groq API. The primary reason for this choice is to avoid the resource-heavy demands of running a local model. By leveraging external computational resources through the Groq API, we can ensure our system remains efficient and scalable without overloading our local infrastructure.

These decisions enabled us to provide high-quality transcription and summarization services without compromising on performance or scalability, aligning with our goal of maintaining an efficient and robust system.

# File Structure

The project directory is organized as follows:

Voc-Notes

├── .git

│ ├── HEAD

│ ├── config

│ ├── description

│ ├── hooks

│ │ ├── applypatch-msg.sample

│ │ ├── commit-msg.sample

│ │ ├── fsmonitor-watchman.sample

│ │ ├── post-update.sample

│ │ ├── pre-applypatch.sample

│ │ ├── pre-commit.sample

│ │ ├── pre-merge-commit.sample

│ │ ├── pre-push.sample

│ │ ├── pre-rebase.sample

│ │ ├── pre-receive.sample

│ │ ├── prepare-commit-msg.sample

│ │ ├── push-to-checkout.sample

│ │ └── update.sample

│ ├── index

│ ├── info

│ │ └── exclude

│ ├── logs

│ │ ├── HEAD

│ │ └── refs

│ │ ├── heads

│ │ │ └── main

│ │ └── remotes

│ │ └── origin

│ │ └── HEAD

│ ├── objects

│ │ ├── info

│ │ └── pack

│ │ ├── pack-9920ce4b580d2a565f1a1355dbf3d99248c5c69e.idx

│ │ └── pack-9920ce4b580d2a565f1a1355dbf3d99248c5c69e.pack

│ ├── packed-refs

│ └── refs

│ ├── heads

│ │ └── main

│ ├── remotes

│ │ └── origin

│ │ └── HEAD

│ └── tags

├── .gitignore

├── .streamlit

│ ├── config.toml

│ └── credentials.toml

├── ADR

│ ├── ADR001.md

│ ├── ADR002.md

│ ├── ADR003.md

│ ├── ADR004.md

│ └── ADR005.md

├── Audio\_chop.py

├── CS615\_Groq\_ClassNote.md

├── CS615\_lecture\_transcript.txt

├── CS615\_recorded\_audio.mp3

├── Data\_tableauhyper.py

├── Engine.py

├── Google\_S2T.py

├── Groq\_LLM.py

├── LICENSE

├── Mongo\_connect.py

├── Note.txt

├── Ollama.py

├── README.md

├── SE577\_lecture\_transcript.txt

├── Voc-Notes-presentation.pptx

├── Voc-Notes\_Documentation-Final.docx

├── \_\_pycache\_\_

│ ├── Audio\_chop.cpython-311.pyc

│ ├── Audio\_chop.cpython-312.pyc

│ ├── Engine.cpython-311.pyc

│ ├── Google\_S2T.cpython-311.pyc

│ ├── Google\_S2T.cpython-312.pyc

│ ├── Groq\_LLM.cpython-311.pyc

│ ├── Groq\_LLM.cpython-312.pyc

│ ├── groq.cpython-312.pyc

│ └── main.cpython-311.pyc

├── app.py

├── pages

│ ├── 1\_record.py

│ ├── 2\_upload.py

│ ├── 3\_generate.py

│ ├── 4\_database.py

│ └── 5\_hyper.py

├── preprocess.py

└── requirements.txt

## Prerequisites

Ensure you have the following prerequisites installed on your system:

* - Python 3.8 or higher  
  - pip (Python package installer)  
  - Git  
  - A Google Cloud account with Speech-to-Text API enabled
* - A groq account for the inference structure

## Step-by-step Installation Guide

Follow these steps to install and set up the project:

1. 1. Clone the repository:  
    >>> git clone https://github.com/invcble/Voc-Notes.git  
      
   2. Navigate to the project directory:  
    >>>cd Voc-Notes  
      
   3. Install the required dependencies:  
    >>>pip install -r requirements.txt
2. 4. Set up Google Cloud Speech-to-Text API, groq API and MongoDB free tier String and set up them as credentials in JSON file.
3. 5. Run the application:  
    >>>streamlit run app.py

# Usage Guide

Once the application is running, you can use the following functionalities:

## Audio Recording

To record audio, navigate to the "Record Audio" section and click the "Start Recording" button. Once you have finished recording, click the "Stop Recording" button. The recorded audio will be saved automatically.

## Uploading Audio

To upload an existing audio file, navigate to the "Upload Audio" section and use the upload button to select your file. The uploaded audio will be processed automatically.

## Generating Notes

After recording or uploading an audio file, navigate to the "Generate Notes" section and click the "Generate" button. The application will process the audio and generate summarized notes, which will be displayed on the screen.

## Downloading or viewing old Notes

After the generation user can choose to see what other Notes are being stored in MongoDB. UI has a option for that, it will redirect to a separate page where users can view and download their generated old notes. New notes upon generation will get automatically synced to the database along with the transcript file.

## Generating .hyper file for Tableau

After recording or uploading an audio file, navigate to the "Data Analysis" section and click the "Generate" button. The application will process the lecture transcript and generate the data file for future data analysis and insight extraction.

# Descriptions of Key Files and Directories

## app.py

The main entry point for the application. It initializes the Streamlit app and defines the different pages and their functionalities.

## Engine.py

Contains core functionalities for the application, including functions for starting and stopping audio recording, processing audio files, and generating notes.

## Google\_S2T.py

Script for interacting with Google Cloud Speech-to-Text API to convert audio recordings into text.

## Audio\_chop.py

Script for processing and segmenting audio files. It prepares the audio data for transcription and summarization.

## pages/1\_record.py

Streamlit script for the audio recording page. It includes functionalities for starting and stopping the recording and updating the UI accordingly.

## pages/2\_upload.py

Streamlit script for the audio upload page. It allows users to upload an audio file and processes the file for transcription.

## pages/3\_generate.py

Streamlit script for the note generation page. It takes the processed audio data and generates summarized notes for the user.

## pages/4\_database.py

Streamlit script for handing MongoDB interactions. It’s mainly responsible for pulling the available notes from the database and downloading them locally.

## Pages/5\_hyper.py

Streamlit script for mainly responsible for generating data storage file for Tableau operations.

**API-key-grok.json**

Configuration file containing API keys for Groq.

**API-key-mongo.json**

Configuration file containing API keys for MongoDB.

**API-key.json**

General configuration file for storing API keys.

**Data\_tableauhyper.py**

Script for creating Hyper files for Tableau data analysis from processed text data.

**Groq\_LLM.py**

Script for interacting with Groq large language models for text summarization.

**Mongo\_connect.py**

Script for connecting to MongoDB and performing database operations.

**Ollama.py**

Script for managing and orchestrating large language models (LLMs) using Ollama.

**preprocess.py**

Script for preprocessing text data, including tokenization and stop word removal, before further analysis.

# Future Work and Improvements

A few future work suggestion that could progress the solution further are,

**Integration of Video Models**

Video Transcription: Capture and transcribe video content for a comprehensive understanding of lectures.

Video Summarization: Extract key frames and create storyboards from video lectures.

OCR and Visual Analysis: Capture and transcribe text from slides and whiteboards using OCR.

**Advanced Data Analytics**

Interactive Visualizations: Develop more advanced Tableau dashboards for trend analysis and keyword tracking.

Real-Time Analytics: Provide immediate feedback and insights during live lectures.

**Improved Summarization Techniques**

Contextual Summarization: Integrate models that better capture lecture nuances.

Topic Modeling: Automatically categorize and summarize lecture topics.

User Experience Improvements

Mobile Access: Develop a mobile version for on-the-go access.

Personalized Dashboards: Offer customized dashboards for tracking and recommendations.

**Integration with Learning Management Systems (LMS)**

LMS Integration: Connect Voc-Notes with systems like Blackboard and Canvas.

Automated Capture: Automatically capture and process lectures from LMS platforms.

Encryption: Ensure data security with advanced encryption methods.

Authentication: Enhance user authentication for secure access.

# References:

* Documentation and developer guides from Google Cloud Speech-to-Text, Whisper AI, PySpark, Groq, Streamlit, and MongoDB.
* https://docs.streamlit.io/
* https://cloud.google.com/speech-to-text?hl=en
* https://github.com/openai/whisper
* https://arxiv.org/abs/2212.04356
* https://console.groq.com/docs/models
* https://spark.apache.org/docs/latest/api/python/index.html
* https://pymongo.readthedocs.io/en/stable/
* Academic papers and articles on speech recognition, data processing, and educational data analysis.
* https://www.researchgate.net/publication/369849536\_Research\_on\_language\_simulation\_and\_speech\_recognition\_based\_on\_data\_simulation\_of\_Machine\_Learning\_System
* https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10346893/
* https://web.stanford.edu/~jurafsky/slp3/