Linguini

Linguini is service to handle video upload & keyword detection.It consists of two parts: A React Native app for User and Node-JS web app for the admin.

System Requirements

- The user-side app to enable the user **recording** a video and upload it on server.Also, the app should provide an **user dashboard** ,rendering the list of videos uploaded by user.
- The web-app for admin to provide an **admin dashboard** rendering the list of videos uploaded by the selected user and functionality to play or download the video.
- To perform **video-processing** and **detect** the keywords in the uploaded video from a dynamic dictionary.
- A built-in video player with **play-from** functionality for every detected keyword.

Solutions Sought for Video Processing

1. Object Detection

   To capture images from videos at the certain frequency and send the unique ones to Amazon Rekognition tool and detect the objects in those frames.

   - **Applicability:**
     - Multiple Object Detection is possible
     - Requirement of manual endeavor to select the objects by user, which is redundant to the very idea of system.

- **Problems with Implementation:**
    - Requires extensive Image-processing in every frame so more internet data-consumption from user side.
    - In order to perform object detection, the application is required to train the machine learning data models with millions of data set to achieve correct confidence level.
    - The data precision will not be sufficient enough to apply it for the real world.

2. Keywords Detection

- **Approach**
  - Extract Audio from the uploaded Video.
  - Generate Text(with correct **time of utterance** of every word) from the audio using some **speech-to-text** conversion tool.
  - Combine the above two to generate the required metadata.

- **Tried/Researched Tools for Speech-to-text**

  The following were tested, as summarized below:

a. Project Oxford-Microsoft

The collection of machine learning offerings is being provided to developers as part of Microsoft's Azure portfolio. Project Oxford is developing in the field of Language Understanding Intelligent Service(LIUS).

Problems:

- It is still in beta version and highly unstable.
- The project website itself recommends not to use for commercial purpose.

b. Facebook's Wit.ai

It sends stream audio to API and gives structured JSON information in return.

Problems:

- Though it has ability to learn from user, it is not considered very stable.
- While testing for Linguini,it was shut down after first 30 sec of use.

c. **IBM's WATSON:**

Watson is developed as an answering system for questions posted in Natural Language.Though it is a very powerful tool and could be perfectly used in this case, the **cost** is $0.01 per minute audio, which is higher than Google Cloud Speech API.

d. **Sensory:**

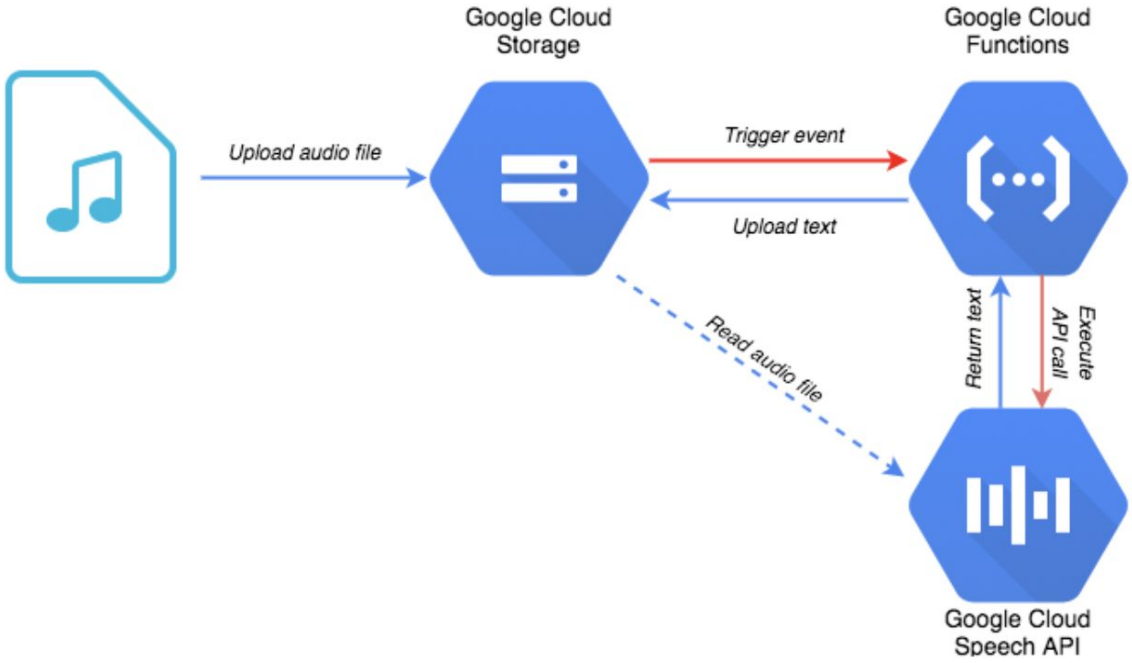
- similar to pocketSphinx
- not free

e. **CMU Sphinx:**

- **PocketSphinx.js:** JS library, runs without flast/plug-in in browser.
    1. Installed PocketSphinx for NodeJS(installation process on README page was broken & dependencies versions out of match-took a while to sync them up)
    2. **model:** 5-pre-alpha
    3. could not process the inbuilt test file
- **Python: pocketsphinx**
    1. installation(very tedious process)
    2. Give a list & detects their occurence
    3. **Problem:** processes only .raw files, mp3/wav to .raw conversion is not supported by any online tool

f. Google Cloud Speech-to-text:

Google Cloud Speech-to-Text enables developers to convert audio to text by applying powerful neural network models in an easy-to-use API. The API recognizes 120 languages and variants to support your global user base.
The user can enable voice command-and-control, transcribe audio from call centers, and more. It can process real-time streaming or prerecorded audio, using Google's machine learning technology.



Cost:

- $0.006 per 15 min audio
- $1.44 (₹ 100)per hour

3. Other Approaches

a. Approach 1:

Play the same video again to give speech input to inbuilt google voice to text in android running in background.
Problem: user will have no choice but to watch the whole video again.

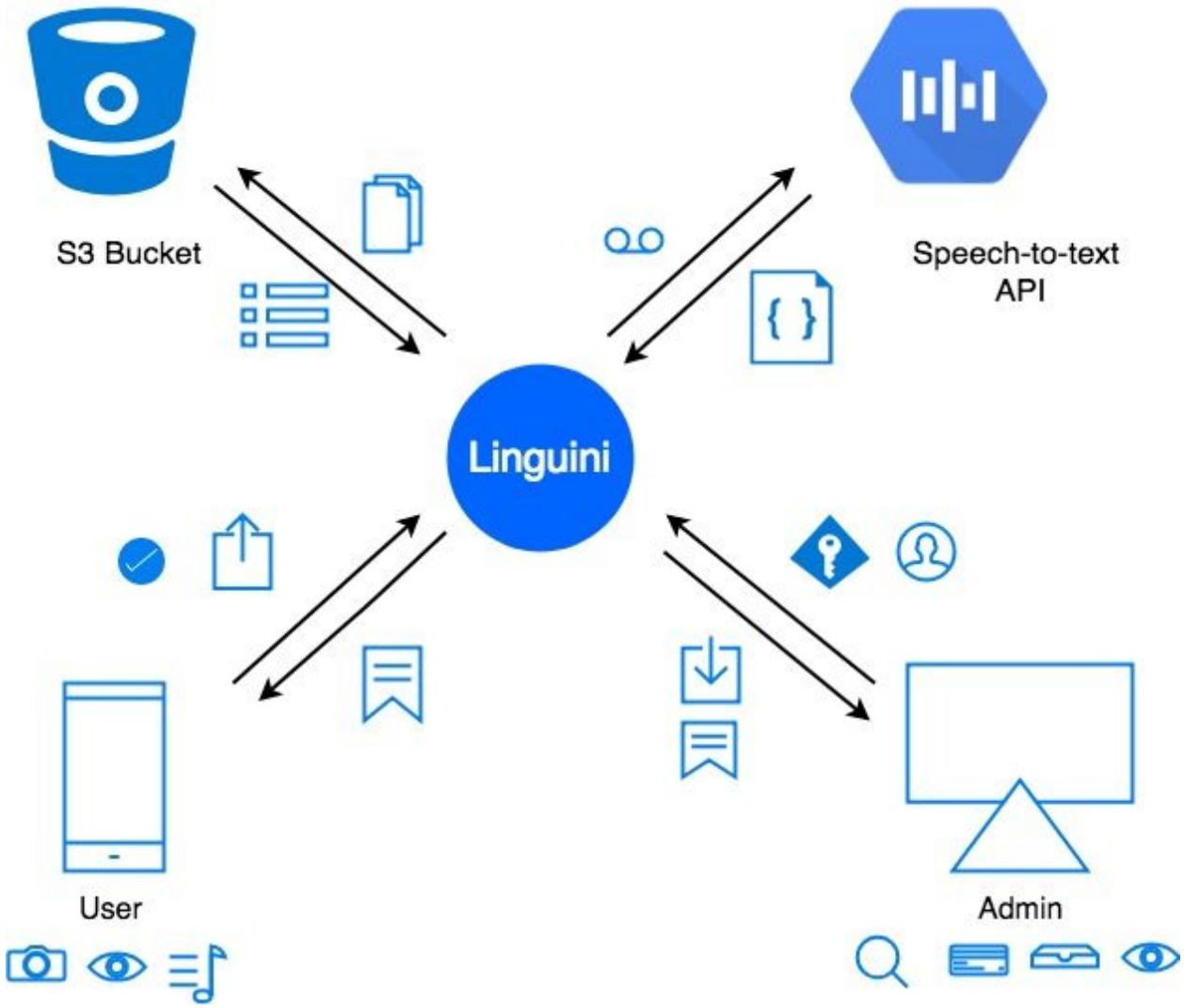
b. **Approach 2:**
   Upload video to youtube, download the captions and then delete the video; using **Youtube Data API**-curl in PHP -used to retrieve youtube resources & data from videos.
   Problem: GET request failed for uploaded videos(not my channel)- 'login credentials invalid'

Final Verdict

Decided to go with Google Cloud's Speech-to-text API.

System Design:

The following diagram describes the basic architecture of the built system, along with the flow of data through various end points:



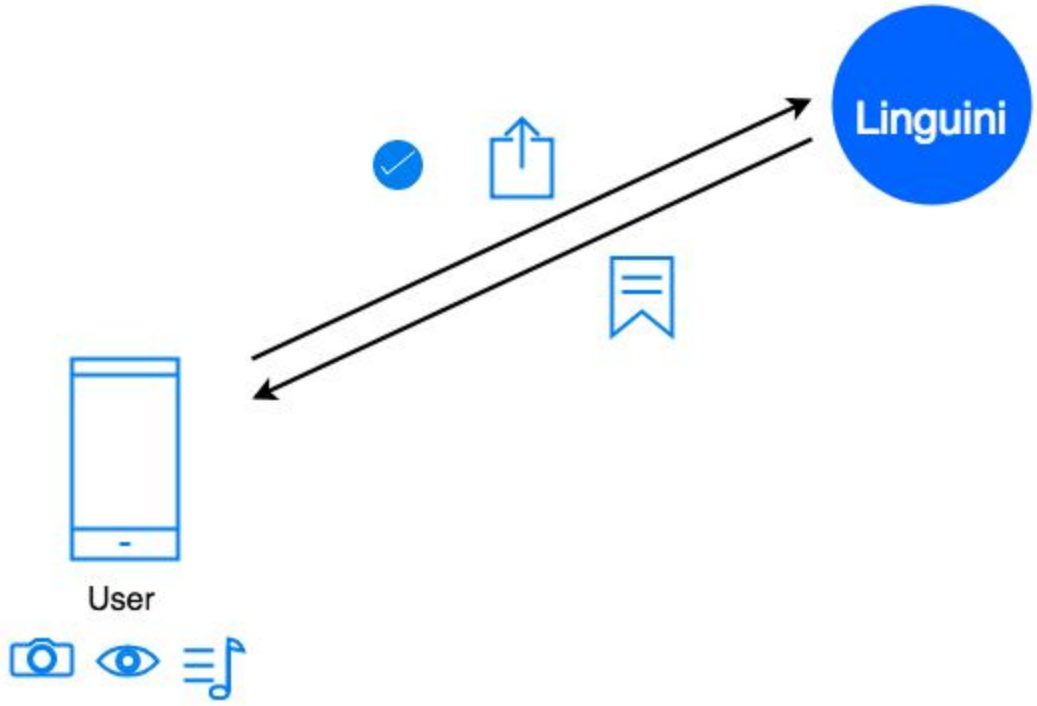
Details of the Components:

1. User App(Android/iOS):

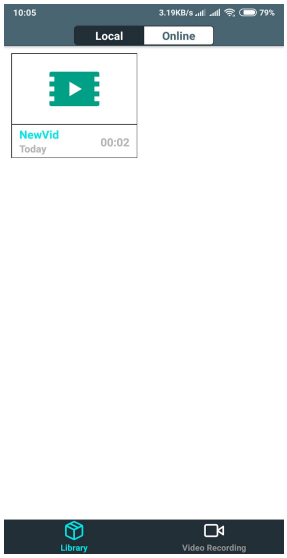
   The User App provides the following functionalities to the user:

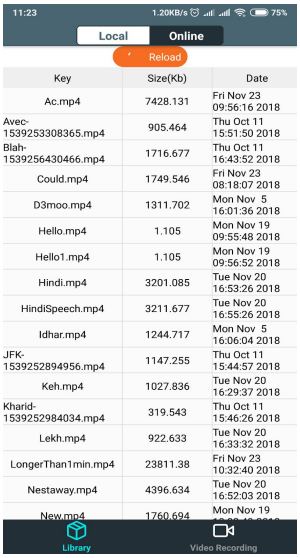
   1. Record a video
   2. Upload the video
   3. Delete the video
   4. Play the video
   5. Get the list of recently videos(available on user's device)
   6. Get the list of uploaded videos

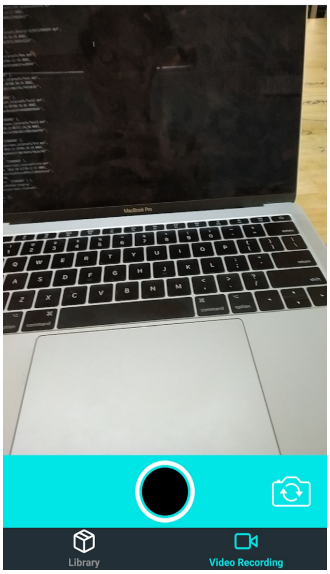
Architecture:



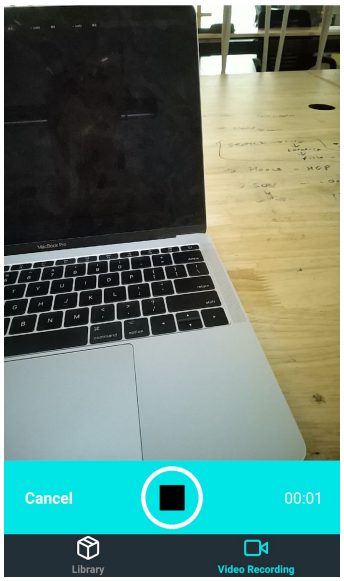
Screenshots:

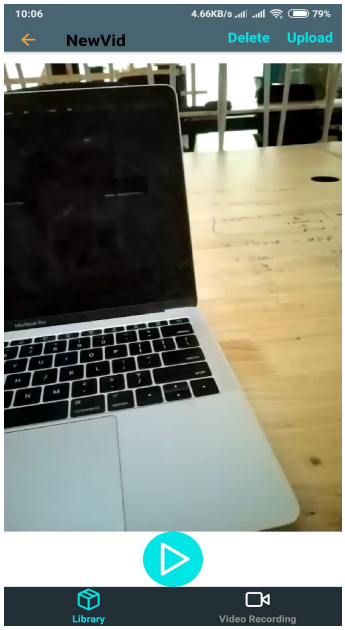






(Screen1: Local Dashboard) (Screen2: Online Dashboard) (Recording Screen: Back Camera)





(Recording the Video) (Naming the Video) (Video Player, with **Delete** & **Upload** buttons)

Demo Video:

Here is the screen recording of the app.<link>

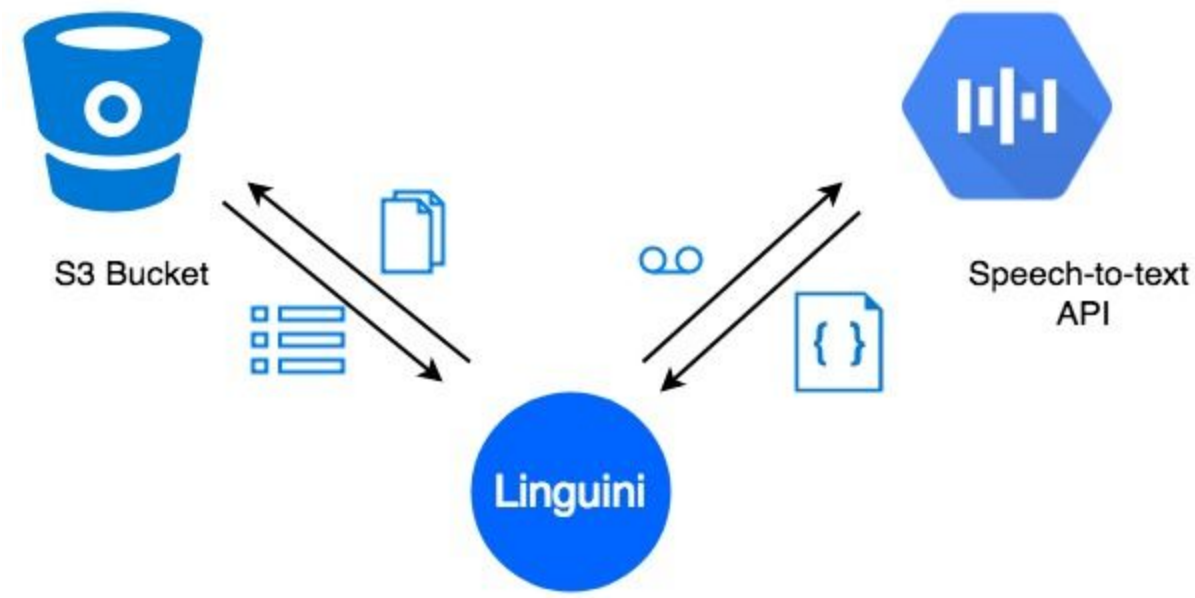
2. Service APIs:

Features:

These are the exposed APIs:

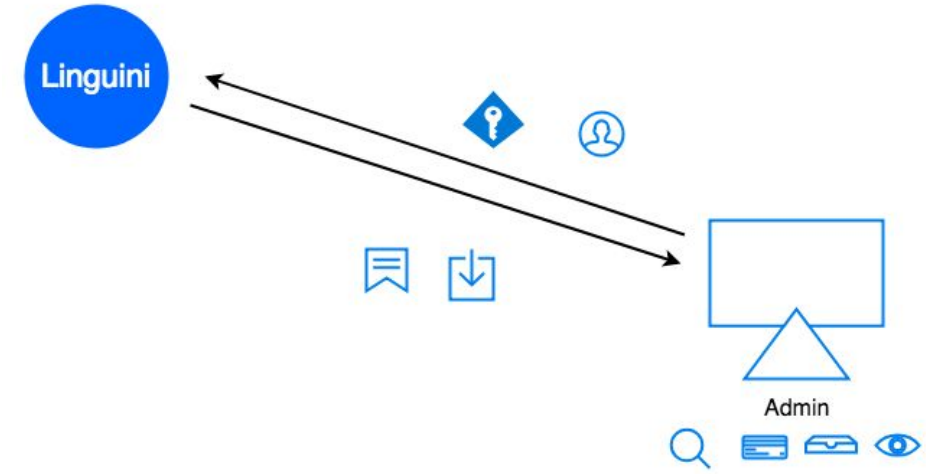
1. **POST:** listing- To generate the list of uploaded videos on Admin dashboard, uses data-tables
2. **GET:** download - To download the clicked video for Admin
3. **POST:** dashboard- To generate the list of uploaded videos for user-side app.
4. **POST:** upload - To upload the video from app to S3 bucket
5. **GoogleCloudSpeechToText**:To extract the text out of video and upload to S3 as **.json** file(with the same name as the video)

Architecture:

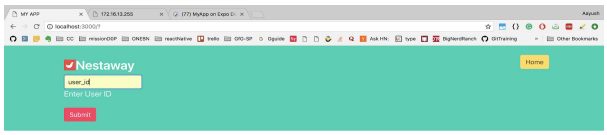


3. Admin Web-App:

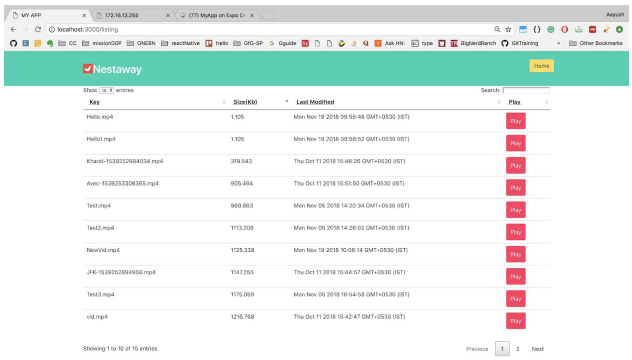
Architecture:



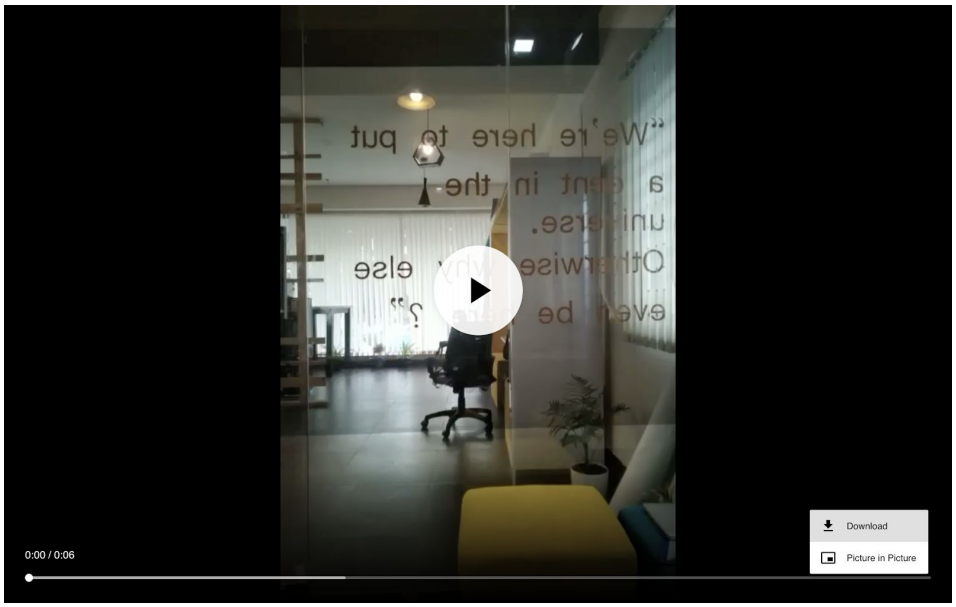
Screenshots:



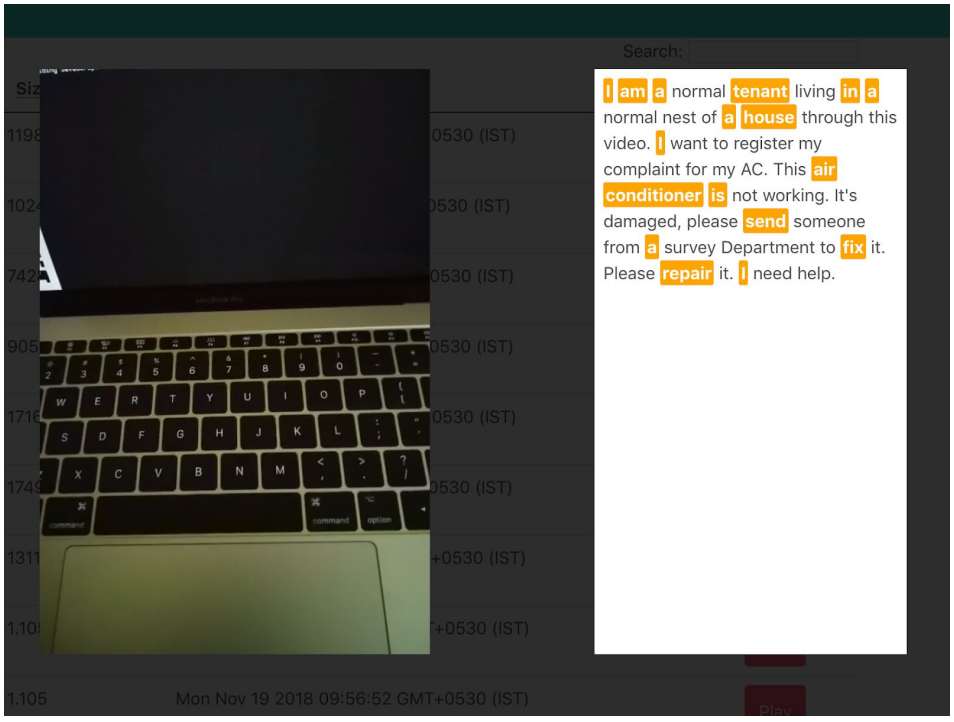
(Screen1: Enter user ID)



(Screen2: Admin Dashboard- list of all the videos uploaded by user)



(Inbuilt Video Player: with **Download** & **Picture-in-picture** options)



(Automatic Word Detection with **Play-From** functionality with every keyword)

Converting Speech-to-text:

Approach:

The process should trigger when the user clicks on the 'Upload' button in app and sends the video to server.

- The server will first convert video(**.mp4**) to audio(**.mp3**) file.
- The audio file is sent to **Google Speech API**
- The API returns a **json** file, containing a list of all the spoken words along with their utterance time.
- This json file should be renamed to the video-title, as given by the user.
- Then both, the video & json file will be uploaded to the server.

Procedure:

- **Video to Audio**
    - **Ffmpeg-extract-audio** node package was used to extract audio from the uploaded video.
- **Audio Encoding**
    - As per the documentation, the audio files requires encoding to the acceptable formats as specified by Google Speech API.
    - Node-lame module was used to convert **.mp3** file to **.wav**
    - **Bitrate** should be kept at **16** while encoding
    - **Model** should be kept at **mono(m):** to encode the double speaker audio to single source audio.
- **Audio Decoding**
    - Decoding requires **.wav** file to be converted to **.flac** format
    - This was done using sox node package.
    - The package requires installation of **SOX-CLI** first, followed by the **npm** package installation.
    - While Decoding, the hertz should be kept at **16k,**otherwise we will get the error `bad hertz rate`.

- **Configuration**
  - The following configuration should be maintained for the above mentioned bitrate and hertz:

```
config = {
    "diarizationSpeakerCount": 1,
    "enableAutomaticPunctuation": true,
    "enableSpeakerDiarization": true,
    "encoding": "FLAC",
    "languageCode": "en-US",
    "model": "video",
    "sampleRateHertz": 16000,
    "enableWordTimeOffsets":true
    };
```