



## Technical Specification

---

Project Title	SUM-UP
Module	CA400 - Final Year Project
Student 1 (Name & ID)	Zubair Asif (18314236)
Student 2 (Name & ID)	Alif Hossain (17314941)
Project Supervisor	Gareth Jones ( <a href="mailto:gareth.jones@dcu.ie">gareth.jones@dcu.ie</a> )
Date of Submission	04/05/2023

# **Table of Contents**

1. Introduction.....	
1.1 Overview.....	
1.2 Motivation.....	
1.3 Glossary .....	
2. Research.....	
2.1 NLP.....	
2.2 Summarisation.....	
2.3 Audio Processing.....	
2.4 Web Development.....	
3. Planning & Time Management.....	
4. Implementation & Design.....	
4.1 System Architecture.....	
4.2 High-Level Design .....	
4.2.1 Data flow Diagram.....	
4.2.2 Application Layout.....	
4.3 Sample code.....	
4.4 Design.....	
4.4.1 Initial Design.....	
4.4.2 Final Design .....	
5. Problems & Resolutions.....	
5.1 Problem 1.....	
5.2 Problem 2.....	
5.3 Problem 3.....	
5.4 Problem 4.....	
5.5 Problem 5.....	
5.6 Problem 6.....	
6. Testing.....	
7. Future Work.....	
8. Installation Guide.....	
9. References.....	

# **1. Introduction**

## **1.1 Overview**

The purpose of our application SUM-UP is to investigate the possibilities of extractive podcast summarising for automated podcast audio summaries. Listeners may choose from a broad variety of topics and genres thanks to the growing popularity of podcasts. However, especially for people with limited time, the length of certain podcast episodes might be daunting. In order to solve this problem, the program creates succinct summaries of podcast episodes. It's crucial to note that our intention is to provide listeners a tantalising look into the essence and highlights of the episode rather than just offering a condensed version of the complete program.

The program uses an extractive podcast summary technique that entails taking important phrases or segments out of the podcast's audio in order to highlight and summarise its essential themes. Extractive summaries are created by choosing and combining significant lines or phrases from the original text. This approach preserves the original text's precise language. Extractive summarization looks at the text to find the most important information, then organises it in a clear and concise manner. These abstracts are accurate, comprehensible, and computationally effective. Extractive summaries are frequently used in news and document summaries, and they may now be used to summarise podcasts. Listeners may rapidly understand the main concepts and takeaways without listening to the full episode thanks to our method, which guarantees that the generated summaries maintain the integrity of the original content.

Users can submit podcast episodes via audio files to the Flask application using our clean and easy to navigate user-friendly interface. The program uses voice recognition algorithms and natural language processing (NLP) techniques to analyse the audio, pick out key passages, and provide succinct summaries. Users may read, listen and download the summaries, enabling them to effectively examine the key points of the audio episodes.

SUM-UP provides a valuable and time-saving alternative for podcast fans, researchers, and professionals who need to keep updated about diverse podcast subjects but have limited time for in-depth listening. It makes it easier to find insightful podcasts and makes them more approachable and palatable. We want to streamline the podcast consuming process by offering attractive audio summaries, allowing consumers to rapidly understand the gist of episodes and decide which ones to delve deeper into.

## **1.2 Motivation**

The idea came to light when we both brainstormed potential ideas that could revolutionise a daily task of ours. Our goal was to create a program that, in a profound way, mirrored our own experiences and offered a steadfast answer to the

problems that troubled us. We consumed numerous hours of audio content each week as passionate supporters of the podcasting genre. We intended to create a tool that would help us choose which podcasts deserved our valuable time and complete focus. Unfortunately, there were times when we became dissatisfied after an apparently interesting program had lasted for 30 minutes. After we had a brief idea we aimed to get some good foundational support and guidance from an experienced supervisor who is knowledgeable on the topic of Summarisation. We proceeded in choosing Professor Gareth Jones.

After careful discussion and consideration we were determined on our idea and began to do market research to see if there was a demand for such a concept application. Our idea played a major role in content discovery (as podcast summaries may be an effective method for finding new material) and decision-making (can help listeners decide whether episodes are worth their time). This only reassured us that it was a good idea to proceed.

### 1.3 Glossary

Python	A high-level, interpreted programming language with a sizable user base, Python is renowned for its straightforward syntax and adaptability. It has a wide range of uses, including artificial intelligence, scientific computing, and web development.
Javascript	JavaScript is a computer language for developing interactive internet applications. JavaScript can power elements like interactive images, carousels, and forms. The language may be used in combination with back-end frameworks like Node.js to support web page mechanics like form processing and payment processing.
Flask	Flask serves as a web framework that makes it simple to create web apps. It is a microframework without an ORM or similar capabilities, with a compact and simple-to-extend core.
Firebase	A platform for building and growing popular apps and games supported by Google and relied upon by millions of companies worldwide.
Frontend	Front-end development is a branch of computer programming that focuses on the coding and creation of website features and functionality that users will see. It is involved with ensuring that the visual components of a website perform properly. Consider the "client-side" of an application to be the front end.
Backend	The backend of a website is made up of servers, programs, and databases. Backend developers provide code that allows browsers to interface with databases and store data in them,

	retrieve data from them, update the data, and delete data or information from them.
Gitlab Repo	A repository is a place where you may save your code and make changes to it. Version control maintains track of your changes.
Summarisation	Summarization is the act of mechanically reducing and rewriting a huge amount of text into a short, concise summary.
Extractive Summarisation	The goal of extractive summarising is to find the important information, which is then extracted and organised together to make a short summary.
NLP	Natural language processing is a branch of linguistics, computer science, and artificial intelligence that studies how computers interact with human language, specifically how to design computers to process and evaluate huge volumes of natural language data.
Machine Learning	The usage and development of computer systems that can learn and adapt without explicit instructions by analysing and drawing conclusions from data patterns utilising algorithms and statistical models
Algorithm	A procedure or set of rules that must be followed in computations or other problem-solving procedures.
Dataset	A collection of connected pieces of information made up of discrete parts but controlled as a whole by a computer.
API	A programming interface allows two or more computer programs to communicate with one another. It is a form of software interface that provides a service to other programs.

## **2. Research**

Before actually implementing the coding for our project, we took the time to examine the specifications for the system's logic as well as the top design concepts for an application such as SUM-UP.

In order for us to create an application such as ours you need a thorough grasp of numerous crucial areas, including Natural Language Processing, Audio Processing, and Web Programming. To further add to the list, one must have good knowledge of Firebase and Docker in order to construct the web application.

## 2.1 NLP

You would need to be familiar with NLP methods including text summarization, named entity recognition, and speech recognition. Additionally, you would need to be knowledgeable with the machine learning techniques that are frequently applied to NLP applications.

## 2.2 Summarization

We had to research and test various summarization techniques in order to find the most compatible and efficient option. Research avenues included sentence scoring, graph based methods, machine learning approaches, latent semantic analysis and rule based methods. This required rigorous research and implementation.

Initially we went with using API'S and pre-made software such as Google Text-to-Speech (gTTS) API and Text Summarization with Latent Semantic Analysis (LSA) algorithm plus PyDub's silence detection.

```
@app.route('/summarize_text_to_speech', methods=['POST'])
def summarize_text_to_speech():
    text = request.json['text']

    # Extractive summarization
    parser = PlaintextParser.from_string(text, Tokenizer("english"))

    ratio = 0.2 # Set your desired ratio
    num_sentences = int(len(parser.document.sentences) * ratio)

    summarizer = LsaSummarizer()
    summary_sentences = summarizer(parser.document, num_sentences)

    summarized_text = " ".join([str(sentence) for sentence in summary_sentences])

    tts = gTTS(summarized_text, lang='en')
    tts.save('output.mp3')
```

We soon realised it was important for us to create our own custom summarizer with NLP, React and python libraries.

## 2.3 Audio Processing

In terms of the audio processing we had to perform research on speech recognition techniques, libraries & tools, audio signal processing, language and linguistics. Only after this research were we able to develop a robust speech recognition system within our application.

## 2.4 Web Development

You would need to be well-versed in web technologies including Flask, Python, HTML, CSS, and JavaScript, as well as having expertise working with web frameworks and libraries like React or Angular. Additionally, you would need to be familiar with database administration and server-side programming. For

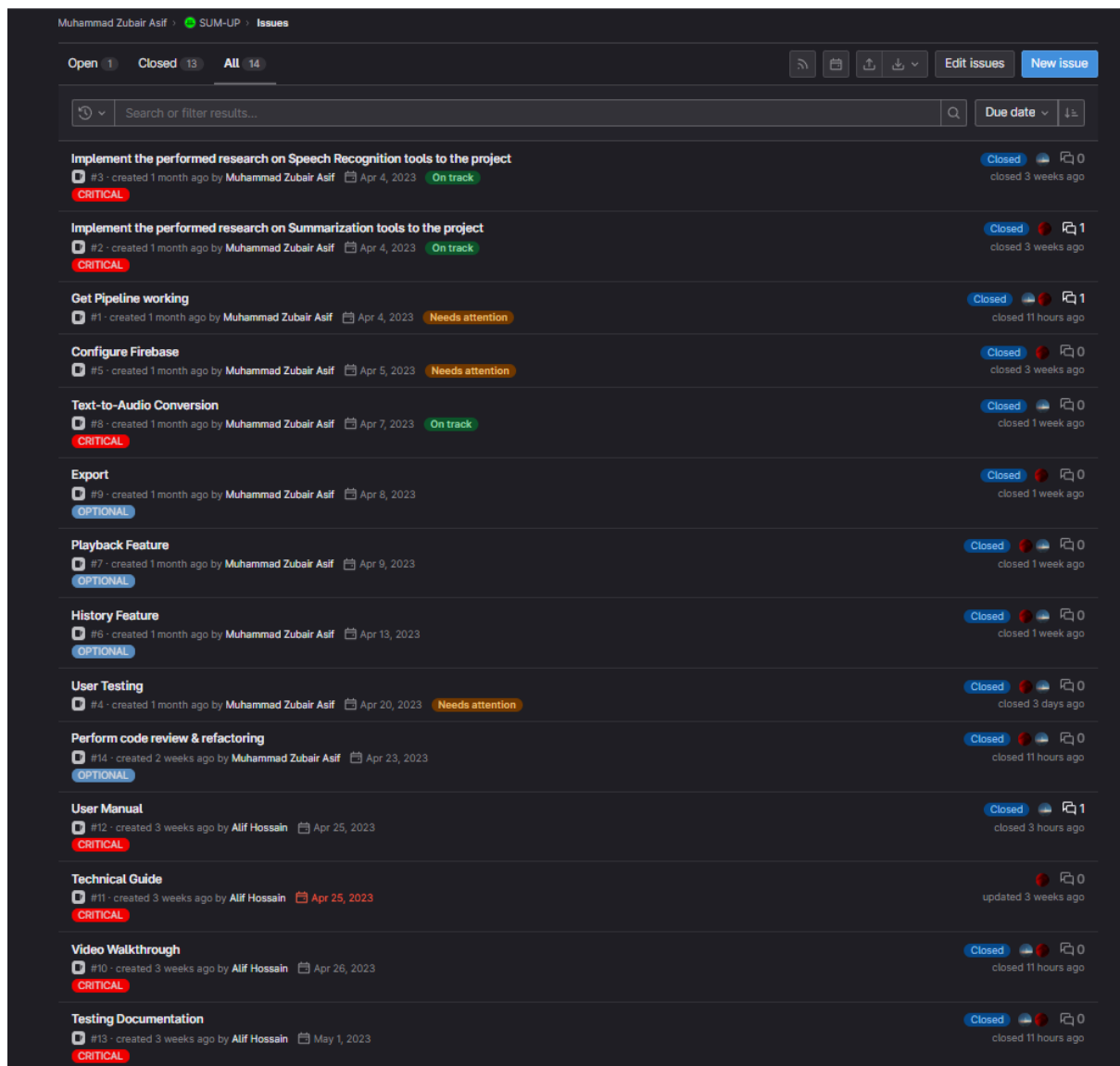
Databases we performed research on various platforms such as MSSQL, PostgreSQL, MongoDB & Firebase.

### 3. Planning & Time Management

In order to stay up to date on all the tasks and different objectives we were tasked with, we utilised the Gitlab issues feature, this feature allowed us to plan and visualise all our goals and milestones that we had to accomplish in order to successfully complete our project.

As you can see from the image below we made use of the priority status, deadlines and the progress status. The benefit of such is that we could easily choose a ticket to tackle and not risk missing or excluding anything of significance.

The ability to assign members aided in understanding the breakdown of work and the balance between the different responsibilities.

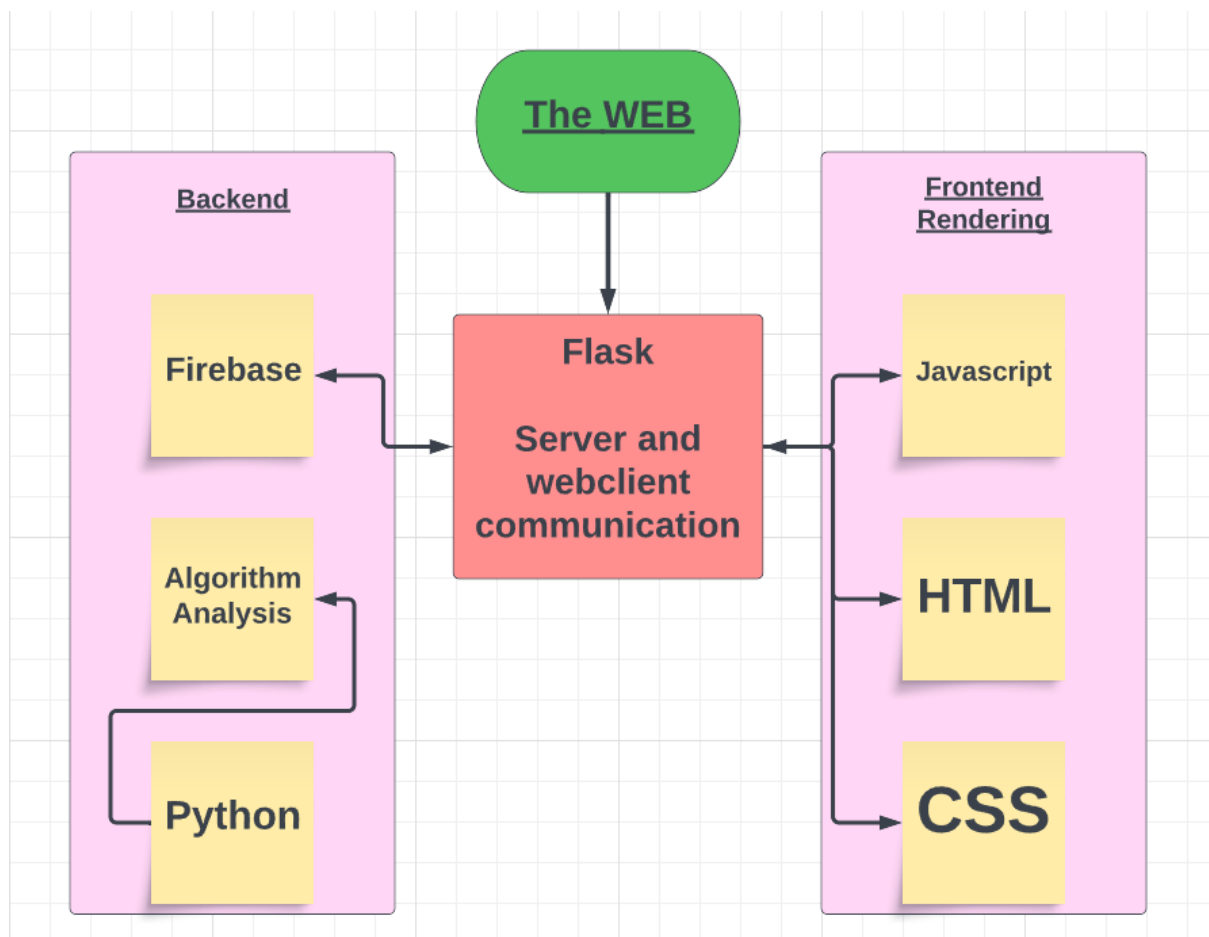


## 4. Implementation & Design

### 4.1 System Architecture

For System Architecture, we have incorporated 4 different programming languages. Python, JavaScript, HTML and CSS were used for both the front end and the back end of the Web Application. We used Google Firebase as our database in order to store data as JSON. This provides us with cross-platform SDKs to help us build and launch our web applications on the web. Flask was used to create the frameworks of the application. We also used Docker to deploy our web application as it provided us with the docker libraries, system tools, code, and runtime.

*System Architecture for SUM-UP Web Application*



Frontend:

Our frontend was written using Javascript, CSS and HTML. We used React framework and libraries from Javascript for our frontend ".js" files. The frontend communicates with Google firebase in order to retrieve authentication about a user as well as their past summary history.



### Backend:

Python was used to create our backend. Since we are highly familiar with the libraries and functionalities of Python, we used it. The processing of the audio files and creation of the summary were done by the backend code. It handled user administration, data storage, text summarization, audio upload and processing, and error handling.

### User Authentication:

We used Google Firebase for user authentication and registering new users. This allows a variety of services such as Firebase authentication and easy-to-use SDKs and ready-made UI libraries for authentication.

### Custom Summarizer:

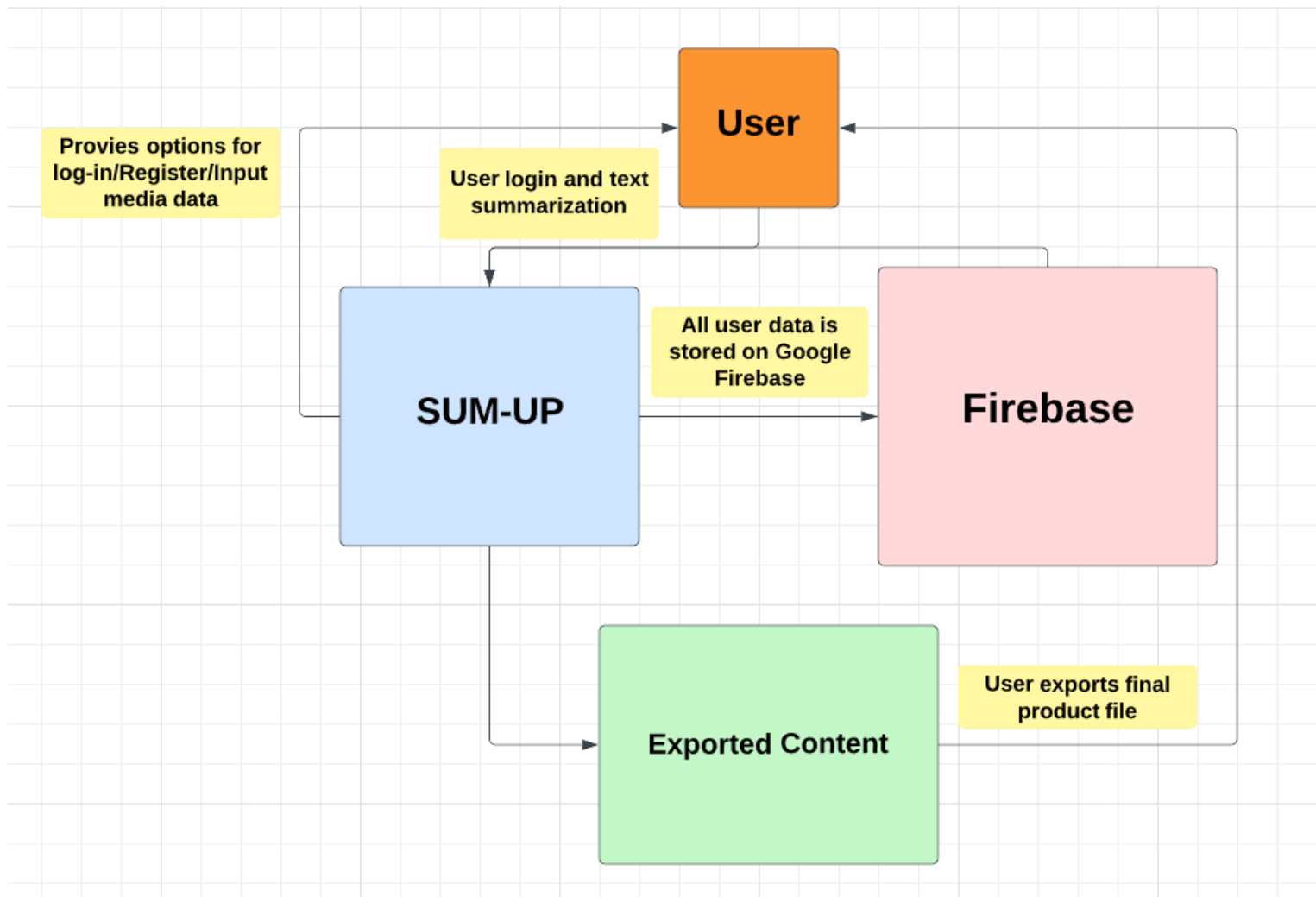
We have built a custom summarizer using Flask and Python libraries. The summarizer takes in text from an audio file after it's been converted. The audio to text convertor is made using a REST API.

### Audio-to-text convertor:

The speech recognition is written in the `celery_app.py` file. It takes in an audio file input and uses SpeechRecognition libraries to transcribe the audio into text. The spaCy library is used to add proper grammar and punctuation to the transcribed text.

## 4.2 High Level Design

### 4.2.1 Data Flow Diagram



### 4.2.2 Application Layout

## User Authentication

SUM-UP

## Login

E-mail

Enter email address

Password

Enter Password

I login

[Already have an account? Sign up](#)

**SUM-UP**

## Signup

**Name\***

**Email\***

**Password\***

**Signup**

[Already have an account? Login](#)

## Homepage

SUM-UP

Choose File No file chosen

Generate text

Initial Text

Summarized Text

Summarize text

0:00 / 0:00

Download as text file Convert to audio file

Run

## Sumarization of an Audio file

# SUM-UP

Choose File

**Initial Text:**

good evening this is the 37th time. I have spoken to you from this office where so many decisions have been made that shaped the history of this nation each time. I have done so to discuss with you some matter that, I believe affected the national interest in all the decisions, I have made in my public life. I have always tried to do what was best for the nation throughout the long and difficult period of Watergate. I thought that was my duty to persevere to make every possible effort the term of office to which you elected me last few days however it has become evident to me but, I no longer have a strong enough political base in the congress testify continuing that effort as long as such a base. I felt strongly that it was necessary to see the process through to its conclusion that to do otherwise the unfaithful to the Spirit of that deliberately difficult process and a dangerously destabilizing precedent for the future but with the disappearance of that base now believe that the constitutional purpose has been served and there is no longer a need for the process to be. I would have preferred to carry through to the finish whatever the personal agony it would have involved and my family you nanimously ur but the interests of the Nation which come before any personal considerations from the disc, I have had with congressional and other leaders. I have concluded that of the Watergate matter. I might not have the support of the congress the. I would consider necessary to back the very difficult and carry

**Summarized Text:**

I must put the interest of America first America needs a full-time and a full-time Congress particularly at with problems we face at home and Abroad. To you to fight through the months ahead for my personal vindication what almost totally absorb the time and attention of both student and the Congress in a period when our or Focus should be on the great issues of Peace abroad and P without inflation at home therefore, I told the nation when in for that office 10 months ago that the leadership of America will be in good hands passing this office to the vice president. I shall resign the presidency effective that noon tomorrow vice president for will be sworn in his president at that hour in this office call the High Hopes for America with which we began term

## History Feature

[illegible]

## 4.3 Code Samples

```
# creating and configuring a celery app
# for background task processing
def create_celery_app(app):
    capp = Celery(
        "bg_worker",
        backend='redis://redis_broker:6379/0',
        broker='redis://redis_broker:6379/0',
        include=["celery_app"],
    )
    capp.conf.update(
        result_expires=3600,
        accept_content=["json", "msgpack"],
        result_serializer="msgpack",
        task_default_queue="celery_queue"
    )
    TaskBase = capp.Task
    capp.app = app

    # task class in order to make sure each task is being executed in
    # allows us to access flask features
    class ContextTask(TaskBase):
        abstract = True

        def __call__(self, *args, **kwargs):
            with capp.app.app_context():
                try:
                    return TaskBase.__call__(self, *args, **kwargs)
                except BaseException as be:
                    raise (be)

    capp.Task = ContextTask
    return capp

# retrieve the status using job id
def get_job(job_id, capp):
    return AsyncResult(job_id, app=capp)

# creates the app
celery_app = create_celery_app(main_app)
```

```
# this task performs audio recog & converts the audio into text
@celery_app.task(name="recognize_audio")
def recognize_audio(file):
    #loading the spacy English model
    nlp = spacy.load("en_core_web_sm")
    #print&log for testing & debugging
    print(f"test: {file}")
    logging.debug(f"test: {file}")
    r = sr.Recognizer()

    with sr.AudioFile(file) as source:
        #opened and recorded for 38secs
        audio = r.record(source, duration=38)

    r.callback = lambda recognizer, audio: print(f"Recognizing audio...")
    text = ""
    try:
        #audio recog using googles speech recog API
        data = r.recognize_google(audio, show_all=False, Language='en-US')
        text += " " + data
    except sr.UnknownValueError:
        print("Could not understand audio")
    chunk_no = 0
    prev_text = ""
    curr_text = ""
    while True:
        chunk_no += 1
        audio = r.record(source, duration=5)
        try:
            data = r.recognize_google(audio, show_all=False, Language='en-US')
            text += " " + data

            current_task.update_state(state="PROGRESS", meta={"text": text})
        except sr.UnknownValueError:
            print("Could not understand audio")

    if len(audio.frame_data) == 0 or r.energy_threshold == 0:
        logging.debug(text)
        doc = nlp(text)
        text_with_punct = ""
        for i, token in enumerate(doc):
            if i > 0 and token.is_alpha and token.is_upper:
                text_with_punct += ". "
            text_with_punct += token.text_with_ws
        print(text_with_punct)
        return text_with_punct
```

The supplied code is a Python script that makes use of the Celery package to set up a background worker for text-to-audio conversion and audio recognition tasks. The code is broken down as follows:

- We create the 'create\_celery\_app' function that configures a celery app for background tasks. In addition to configuring additional options like result expiration, allowed content types, result serialisation, and the default task queue, it establishes the backend and broker URLs for Celery to use (Redis).
- The 'ContextTask' class within the 'create\_celery\_app' extends the 'Task' class which basically makes sure that tasks which are executed in the Flask app's context, giving all of its features. To provide Flask-specific features, it encapsulates the job execution within a Flask app environment.
- It is possible to access the status of a job or task using the Celery AsyncResult class by utilising the 'get\_job' function.

- Calling the 'create\_celery\_app' function with the main Flask app as an input creates the 'celery\_app' variable. The Celery app is now ready for use.
- The '@celery\_app.task' decorator is used to define the 'recognize\_audio' task. This process turns audio data into text and conducts audio recognition. To load and process audio files, it uses the SpeechRecognition library. The audio is recognized and converted into text using the Google Speech Recognition API. Then the identified text is given back.
- The Spacy English model is loaded into the 'recognize\_audio' job for further text processing. Using the SpeechRecognition library, the audio file is opened and 30 seconds are captured. The text that was detected in the audio is then added to the text variable once the audio has been processed using the Google Speech Recognition API.
- To accommodate lengthier audio files, further processing is done in 5 second segments. The Google Speech Recognition API is used to identify the chunks, and the detected text is added to the text variable. 'Current\_task.update\_state()' is used to update the task's status to reflect the text currently being processed
- The last identified text is processed using the Spacy NLP package when there is no more audio data available or when the energy threshold is met. The text is enhanced by punctuation, which divides sentences based on capitalization. The task's output is printed and returned together with the processed text.

The significance of the following code is that it explains how to use background workers in a Flask project to perform time-consuming activities asynchronously. This is vital for our project as without it there would be major delays and it would impact the responsiveness of SUM-UP

```
def text_summerizer(text):
    nlp = spacy.load('en_core_web_sm')
    punctuations = '!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~\n'

    doc = nlp(text)

    # filter out stop words and punc
    sentences = [sent for sent in doc.sents if not all(token.is_stop or token.text in punctuations for token in sent)]
    words = [token.text for sent in sentences for token in sent if not token.is_stop and token.text not in punctuations]

    # Calculate word frequencies
    word_frequencies = {}
    for word in words:
        word = word.lower()
        if word not in word_frequencies:
            word_frequencies[word] = 1
        else:
            word_frequencies[word] += 1

    # Normalise word frequencies
    max_frequency = max(word_frequencies.values())
    for word in word_frequencies:
        word_frequencies[word] /= max_frequency

    # Calc sentences scores
    sentence_scores = {}
    for sent in sentences:
        sentence = sent.text.lower()
        sentence_scores[sent] = sum(word_frequencies.get(word.text.lower(), 0) for word in sent)

    # Choose top sentence for summary
    summary_size = min(3, len(sentences))
    summary_sentences = nlargest(summary_size, sentence_scores, key=sentence_scores.get)
    summary = ' '.join(sent.text for sent in summary_sentences)

    return summary
```

A function named 'text\_summarizer' that extractively summarises a text is defined in the supplied code. The code is broken down as follows:

- We first use `spacy.load('en_core_web_sm')` to load the English language model from the spaCy library. This approach allows us to analyse text and gives linguistic annotations.
- Establish a string variable for punctuation that includes a selection of punctuation letters as well as the newline character.
- Use the loaded spaCy model to process the supplied text: `'doc = nlp(text)'`. By breaking up the text into sentences and words, this tokenizes the text and gives each token linguistic characteristics.
- Remove any stop words and punctuation from the sentences, then add the sentence lists with the filtered sentences. Additionally, gather all non-stop words and tokens without punctuation from the words list. By iterating over the word list, determine the word frequencies.
- The frequency values are kept in the `word_frequencies` dictionary, which has unique words as keys and frequency values as values.
- By dividing each frequency by the highest frequency in `word_frequencies`, normalises the word frequencies. By doing this, the frequencies are guaranteed to be inside a normalised range.
- By adding the normalised frequencies of the terms in the sentence, get the score for each sentence. The `sentence_scores` dictionary, which has sentence objects as keys and matching scores as values, is where the sentence scores are kept.
- Pick the strongest sentences to use as the summary. With a maximum of three sentences or the number of sentences, whichever is less, the `summary_size` variable controls how many sentences should be included.

The reason for specifically mentioning the above code as it plays a significant role in the operation of our application. It offers the ability to perform extractive summarisation within the NLP domain space.

```

JS Audio.js X
text_summerizer-main > frontend > src > components > Audio > JS Audio.js > AudioToTextConverter
1 import React, { useContext, useEffect, useState } from "react";
2 import styles from "../Audio.module.css";
3 import { Button, Spin, notification } from "antd";
4 import { BASE_URL } from "../../constants";
5
6 import { addDoc, collection, doc } from "firebase/firestore";
7 import { db, auth } from "../../firebase";
8 import { AppContext } from "../../App";
9 import TextArea from "antd/es/input/TextArea";
10
11 function AudioToTextConverter() {
12   const {
13     file,
14     setFile,
15     generatedText,
16     setGeneratedText,
17     summarizedText,
18     setSummarizedText,
19     loader,
20     setLoader,
21   } = useContext(AppContext);
22
23   const currentUser = auth.currentUser;
24
25   const { uid, email } = currentUser;
26
27   const handleInputFile = (e) => {
28     setFile(e.target.files[0]);
29   };
30
31   const handleGeneratedTextAreaChange = (value) => {
32     setGeneratedText(value);
33   };
34
35   const handleSummarizedTextAreaChange = (value) => {
36     setSummarizedText(value);
37   };
38
39   const saveDataToFirebase = async () => {
40     if (
41       [null, "", undefined].includes(generatedText) ||
42       [null, "", undefined].includes(summarizedText)
43     ) {
44       notification.warning({ message: "Please fill the text field" });
45       return;
46     }
47     setLoader(true);
48     notification.success({ message: "Saving data to the firebase" });
49     try {
50       const audioToTextCollectionRef = collection(db, "audioToText");
51       const audioToTextDataRef = doc(audioToTextCollectionRef, uid);
52       const audioToTextSubCollectionRef = collection(
53         audioToTextDataRef,
54         "records"
55       );

```

```

54       "records"
55     );
56     const data = {
57       uid,
58       email,
59       generatedText,
60       summarizedText,
61     };
62
63     await addDoc(audioToTextSubCollectionRef, data);
64     setFile("");
65     setGeneratedText("");
66     setSummarizedText("");
67     notification.success({ message: "Saved Successfully!" });
68
69     setLoader(false);
70   } catch (e) {
71     setLoader(false);
72     console.error("Error adding document:", e);
73   }
74 };

```

This AudioToTextConverter() function from the 'audio.js' file within our Frontend code was used to save data to Firebase Firestore.

- It defines the AudioToTextConverter functional component. It imports needed libraries and constants such as React, useContext, useState, Spin, Button, notification, TextArea, BASE\_URL, addDoc, collection, doc, database, and auth into the component.
- The AppContext object is produced by the createContext function. It is used to maintain the application's state, which contains attributes such as file, generatedText, summarizedText, and loader.
- HandleInputFile, handleGeneratedTextAreaChange, and handleSummarizedTextAreaChange are some of the functions defined by the component to handle user interactions.
- When you click the save button, the saveDataToFirebase method is called. It first checks to see if the generatedText and summarizedText fields are not empty, and if they are, it displays a warning signal. If the fields are not empty, the loader is set to true, a new document is created in the audioToText collection with the user ID as the document ID, a new sub-collection named records is added to the new document,

and a new document is created inside it with the uid, email, generatedText, and summarizedText fields as data. Then it clears the input fields, disables the loader, and shows a success message.

## 4.4 Design

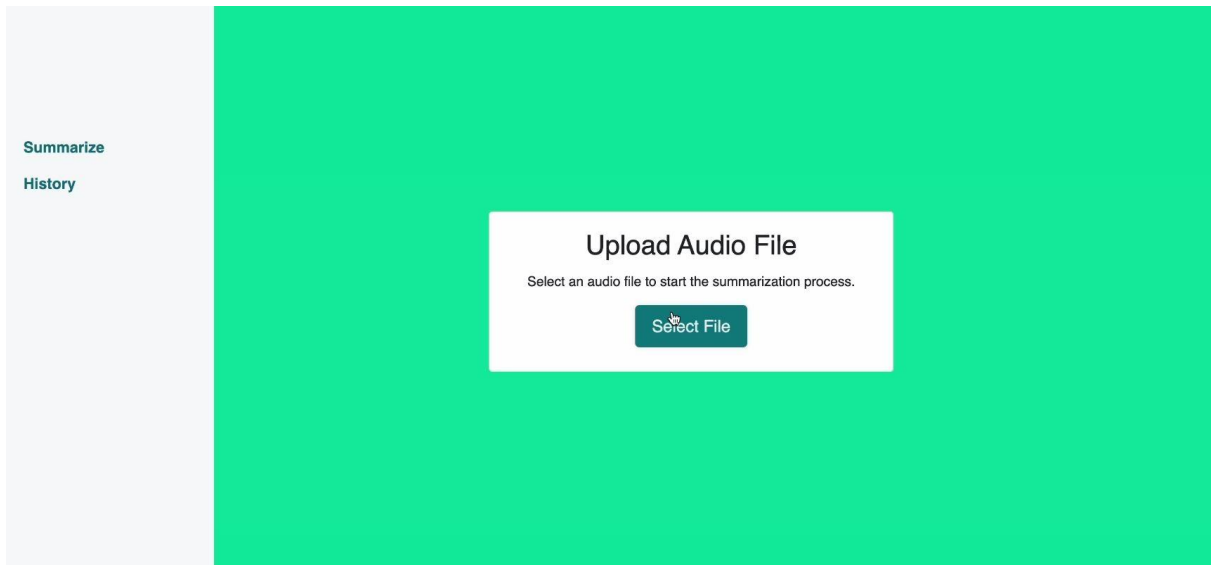
### 4.4.1 Initial Design

Below you will find our initial design. The first screen that the user will be presented with. As you can see there is not a lot that the user is presented with. If the user would like to perform any functions they must navigate to the nav bar on the left and choose an option.

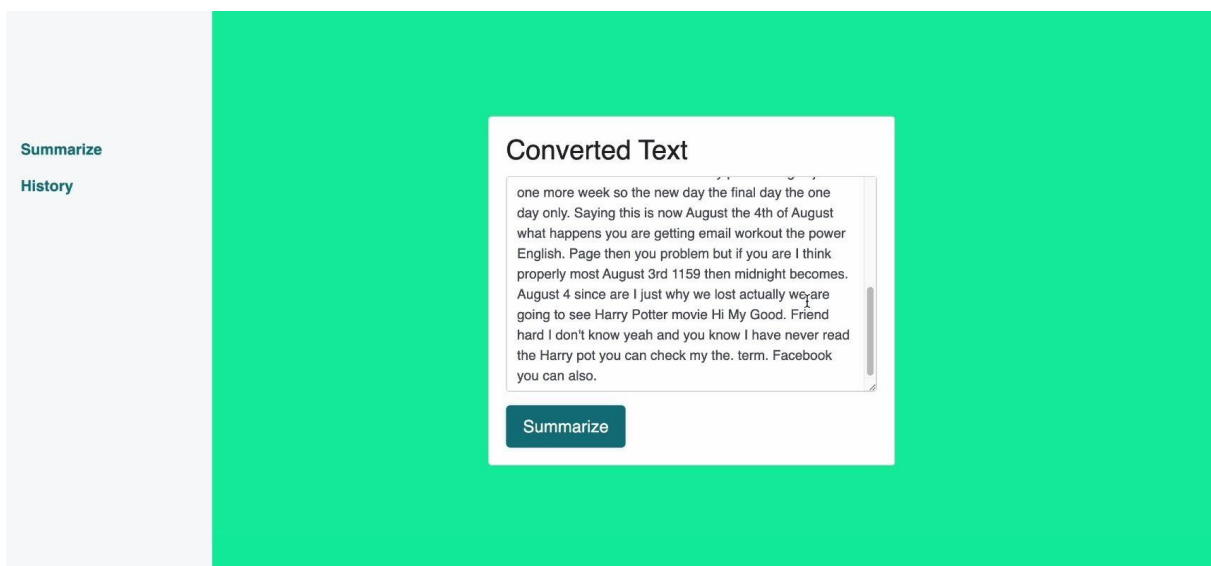


Below you will find the screen that appears after the user selects the “Summarize” option. The user will be presented with a box with the option to upload an audio file.

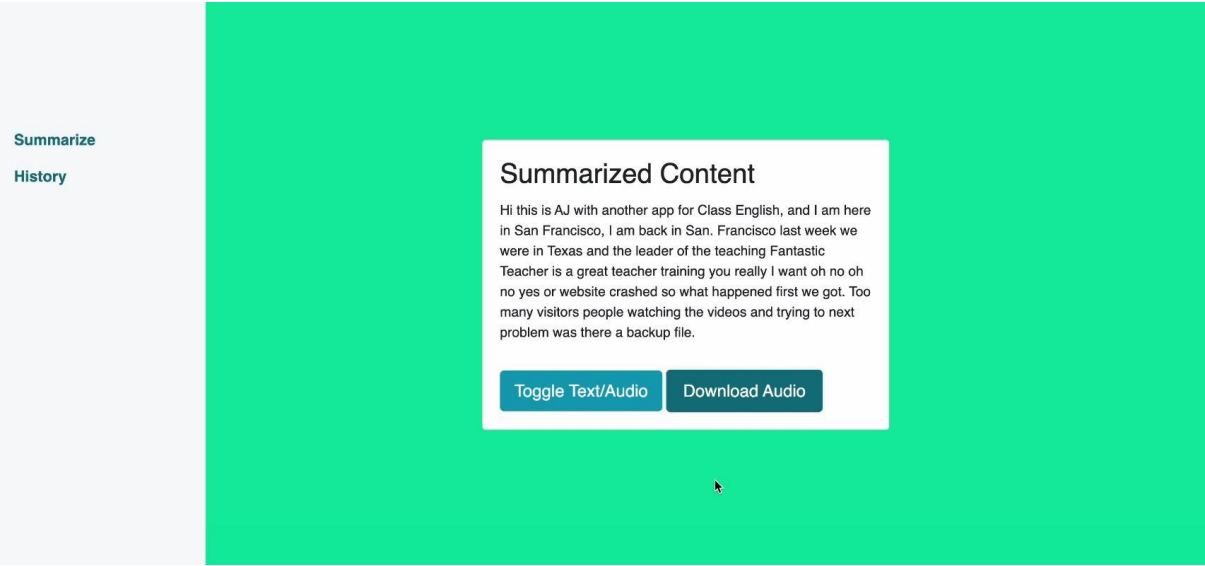




Below you will find the screen which you are presented with once the audio file has been transcribed successfully. You will be able to view all the data with the option to then proceed to the summarisation step by clicking on the button.



Below you will find the screen which you are presented with once the data has been successfully summarised. The user has the option to toggle and view the audio version of the summary and download the file or they can opt to download the text file into a word doc format.



Below you will find the screen you are presented with when you click on the History option on the left hand side of the screen on the white nav bar. This will take you to a screen where you can view past generated text, summaries and their timestamps.

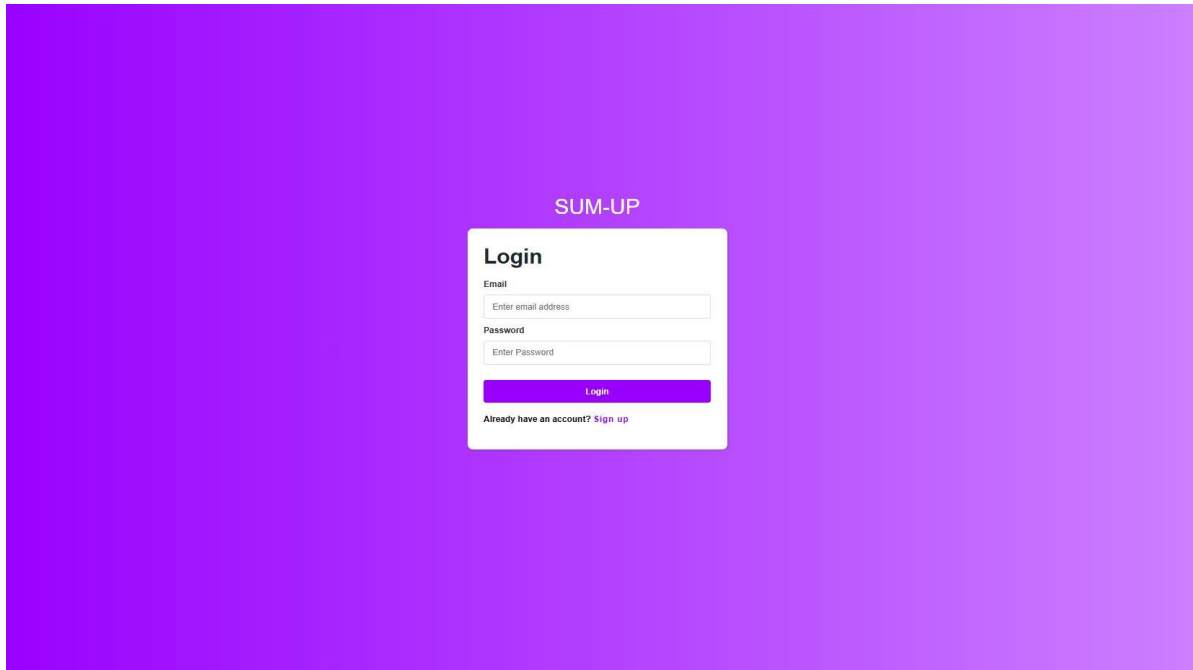
		History		
		Timestamp	Generated Text	Summarized Text
Summarize	History	20/01/1970, 15:55:43	Hi this is AJ with another app for Class English, and I am here in San Francisco, I am back in San. Francisco last week we were in Texas and the leader of the teaching Fantastic Teacher is a great teacher training you really I want oh no oh no yes or website crashed so what happened first we got. Too many visitors people watching the videos and trying to next problem was there a backup file. Our website backup had a problem the website hosts so we	Hi this is AJ with another app for Class English, and I am here in San Francisco, I am back in San. Francisco last week we were in Texas and the leader of the teaching Fantastic Teacher is a great teacher training you really I want oh no oh no yes or website crashed so what happened first we got. Too many visitors people watching the videos and trying to next

Overall Comments:

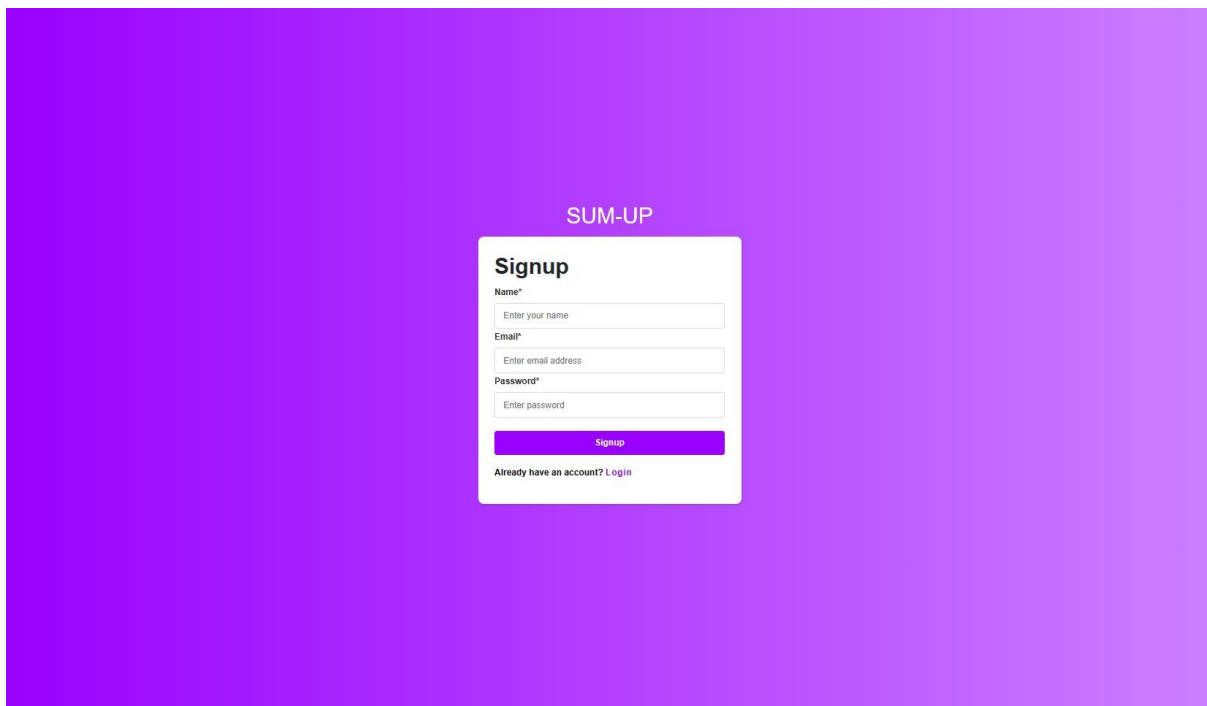
The initial design was not aesthetically pleasing. We felt that the colours were off and not the most pleasing to look at. We had a few comments made by friends and family stating the main functions were hidden and hard to really find. It gave a cumbersome and poorly designed first impression, which was not something that we were hoping to give. Furthermore, there were too many different pages/screens in order to navigate through the application to perform the summarisation.

#### 4.4.2 Final Design:

Below is the screen the user is presented with when they first visit the application. They will be presented with a clear, concise, well designed Login or Signup box, where they can enter their credential and proceed. The first thing you may notice from the new design is that the colour has changed from a blinding lime green to an imperial purple tone.



The image shows a login form centered on a solid imperial purple background. At the top, the text "SUM-UP" is displayed in a light purple, sans-serif font. Below it, the word "Login" is written in a bold, black, sans-serif font. The form contains two input fields: "Email" with the placeholder text "Enter email address" and "Password" with the placeholder text "Enter Password". Both fields have a light purple border. Below the password field is a solid purple button with the word "Login" in white, sans-serif font. At the bottom of the form, the text "Already have an account? Sign up" is displayed in a small, black, sans-serif font, with "Sign up" being a link.

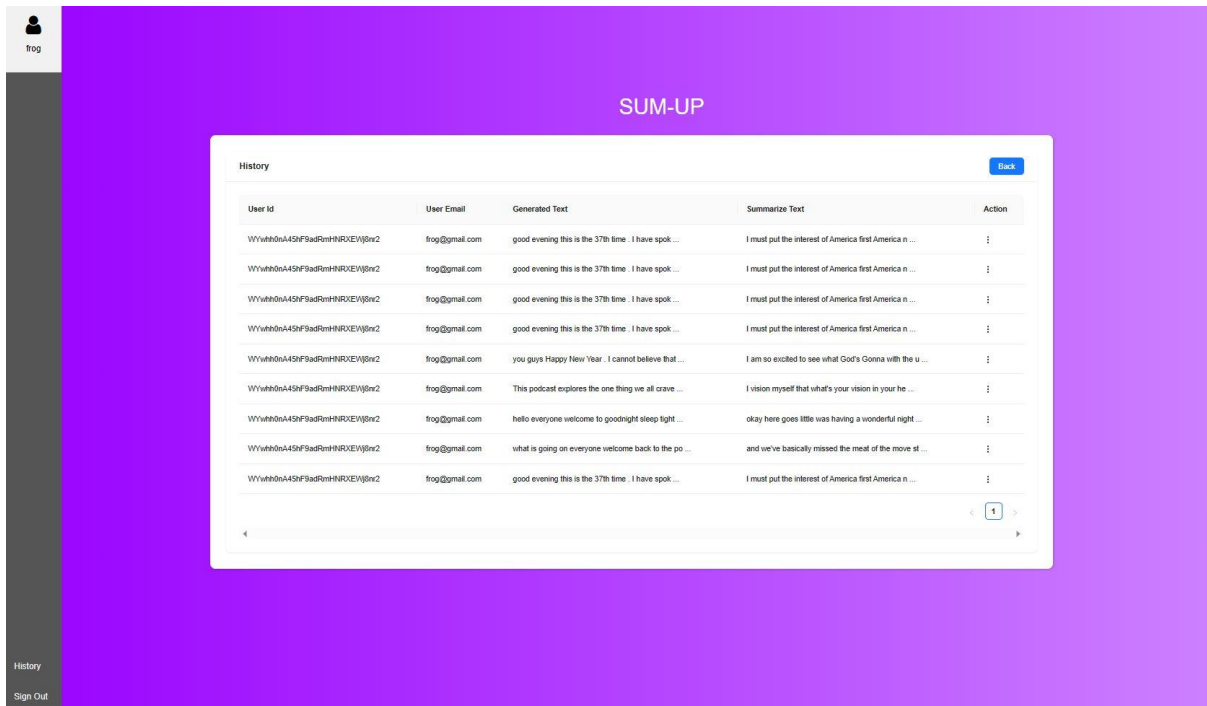


The image shows a signup form centered on a solid imperial purple background. At the top, the text "SUM-UP" is displayed in a light purple, sans-serif font. Below it, the word "Signup" is written in a bold, black, sans-serif font. The form contains three input fields: "Name\*" with the placeholder text "Enter your name", "Email\*" with the placeholder text "Enter email address", and "Password\*" with the placeholder text "Enter password". All fields have a light purple border. Below the password field is a solid purple button with the word "Signup" in white, sans-serif font. At the bottom of the form, the text "Already have an account? Login" is displayed in a small, black, sans-serif font, with "Login" being a link.

Below is the screen the user is presented with when they successfully enter the application after the authentication screen. The user is presented with a compact and concise arrangement of features. We maintained the structural design of the previous version such as having the navigation options on the left hand side and the main operational features in the centre of the screen. Unlike the previous version where the user would be directed to a new screen after each step such as audio to text then summarization, with this design the user is able to view all of that on the current screen.



Below is the screen the user is presented with when they visit the History section by clicking on the history option on the bottom left corner of the screen. As you can see from the image below this is significantly different than the previous version as it is more legible, clearer in detail with more information about the user including ID and Email, along with a simpler design. This prevents the user from getting an information overload.



Below is the screen the user is presented with when they successfully perform the summarisation. See from the image below, the user has the ability to view everything on a single screen including the initial text. Andy summarised text. This is quite different compared to the last application. Furthermore, the user does not need to toggle. To view the audio component.



Overall Comments:

Our final design contained multiple major changes and improvements compared to the initial design. The first key feature we implemented was the ability to view everything in the central screen / home page. The second major change was the History page where there is now increased visibility and features. Some minor changes included adding a home button along with the username on the top left of the navigation bar, along with the colour and font changes.

## **5. Problems and Resolution**

### **5.1 Handle large files**

We encountered an issue where the size of the audio files was an issue for the application to execute properly. Additionally it was taking too long to perform.

```
@celery_app.task(name="recognize_audio")
def recognize_audio(file):
    nlp = spacy.load("en_core_web_sm")
    print(f"test: {file}")
    logging.debug(f"test: {file}")
    r = sr.Recognizer()
```

**Solution:** We resolved this issue by creating a Celery file that takes in an audio file as input and uses SpeechRecognition libraries to transcribe the audio into text. Then uses spaCy library to add proper grammar and punctuation to the transcribed text.

### **5.2 Normalising words**

```
# Normalize word frequencies
max_frequency = max(word_frequencies.values())
for word in word_frequencies:
    word_frequencies[word] /= max_frequency
```

**Solution:** We were able to resolve this by creating a max\_frequency function.

### **5.3 Audio to text misses dots**

**Solution:** in general any new capital character comes it will put dot in previous word.

### **5.4 Communicate progress to the backend, so the user interacts**

**Solution:** Solution we did is we are sharing chunks output along with previous output.

### **5.5 Having issues with converting the resulting summary into an audio file.**

```
@api.route('/doc/export')
class DocumentExportResponse(Resource):
    """Class for Document text audio export"""
```

**Solution:** We resolved this problem by creating "DocumentExportResponse" function and coding the data(.json) to be converted to audio.

```
@validate_form_data(text_upload)
def post(self):
    """Endpoint for text summarization"""
    # define the text to convert to audio
    text = request.json['text']
    doc = docx.Document()
    doc.add_paragraph(text)
```

### 5.6 Our initial idea of using API's to create a summarizer didn't work.

**Solution:** Our initial idea was to create the summarizer using apis but we felt that it would not be able to handle large files with background/white noise well. We also felt that it lacked ingenuity, hence we decided to create our own customer summarizer using Natural Language Processing alongside react and python libraries. This allowed us to have a summarizer capable of handling large files with low quality audio.

## 6. Testing

Please refer to our testing documentation located below for more details on the testing process.

Link:

[https://gitlab.computing.dcu.ie/asifm2/2023-ca400-asifm2-hossaia5/-/blob/master/docs/documentation/Testing\\_SUM-UP.pdf](https://gitlab.computing.dcu.ie/asifm2/2023-ca400-asifm2-hossaia5/-/blob/master/docs/documentation/Testing_SUM-UP.pdf)

### Heuristics Testing:

The process of heuristic testing involves specialists using general guidelines to assess the usability of user interfaces during independent walkthroughs and indicate problems. Evaluators make use of well-known heuristics and provide insights that might aid design teams in improving the usability of products from the very beginning.

We undertook Heuristics testing/evaluation in order to optimise usability and at the same time eliminate design deficiencies.

### User-Testing:

We conducted user testing in the DCU labs in order to test and learn more about our web application from others. We asked 4 unanimous students from our year to use our application. They did this by following a user manual created to help them

download and navigate through our application. They each used the application with a variety of audio files each of different lengths. After which they were able to see the application running and note its features.

After completing the testing each of the students were given a consent form and a survey to complete. They were asked a variety of questions each of which were relevant for us to find out more about our application, how we can further improve it, if they found our application useful or not, would they recommend it and for what they would use it for. After which the user test was completed.

We were able to use the data collected from the user testing to write a testing document. Which showcased our findings through writing, diagrams, pie charts and histograms. This provided us with information on how to further develop our application and aspects to fix.

## **7. Future Development**

Upon the completion of our user testing we received some feedback on what improvements could be made to our web application. Below are some of the improvements that were suggested and also some that were not possible to finish due to the time-frame.

- Potential Idea: Adding a video summarization feature to our app that would summarise video file content. This is something that we had in mind but were advised to not focus on this as it had significant technical difficulties and would not be completed within the time frame.
- Another feature that was suggested through user testing was creating a more aesthetically pleasing User-interface. As we concentrated more on the novelty aspect of our project we left the User-interface design for last. If given more time we would like to focus on adding animation and creating a design layout for the user-interface.
- Lastly we would like to incorporate more audio file formats that can be summarised. Currently the application only accepts .WAV files, this can be a hindrance for those who want to use other file types such as mp3 or ogg. Being able to support different formats can help users summarise and use our application for more material.

## **8. Installation Guide**

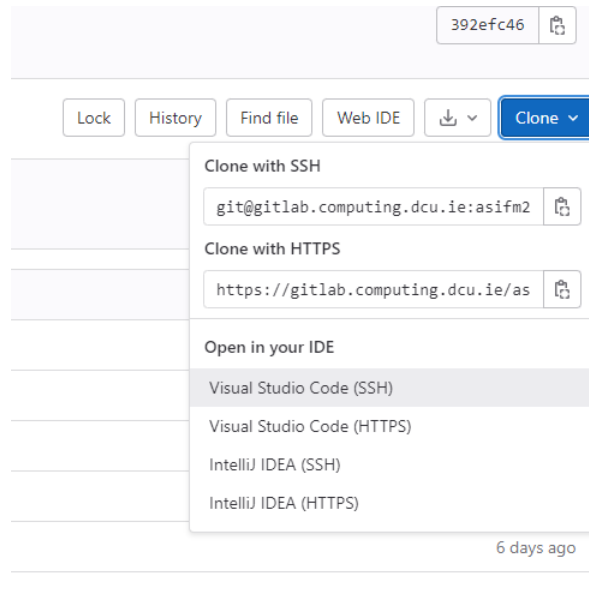
Step 1: Go to the gitlab repository:

<https://gitlab.computing.dcu.ie/asifm2/2023-ca400-asifm2-hossaia5/-/tree/master/src>

Step 2: Click the Clone button and use the clone with HTTPS.



Step 3: Use git clone to download the web app by using command "git clone <https://gitlab.computing.dcu.ie/asifm2/2023-ca400-asifm2-hossaia5.git> " in your command prompt on windows or terminal on any unix based machine.



Step 4: Build Frontend

You will need to build frontend by using the command (within the frontend directory)

**yarn install**

**yarn build**

Step 5: Build Docker

You will need to build docker using the following command (must exit frontend directory)

**docker-compose build**

**docker-compose up -d**

Step 6: Access the Application

You can view the app running on the following link

**Localhost:5005**

## **9. References:**

- Audio Summarization for Podcasts-IEEE, Aneesh Vartakavi; Amanmeet Garg; Zafar Rafii <https://ieeexplore.ieee.org/document/9615948>
- PodSumm: Podcast Audio Summarization, ANEESH VARTAKAVI and AMANMEET GARG, Gracenote Inc <https://arxiv.org/pdf/2009.10315.pdf>
- Deep Learning for NLP and Speech Recognition, Uday Kamath , John Liu , James Whitaker <https://link.springer.com/book/10.1007/978-3-030-14596-5>

- Speech Recognition in Python: <https://stackoverflow.com/questions/12239080/getting-started-with-speech-recognition-and-python>
- For Docker: <https://docs.docker.com/>
- Javascript: <https://www.javascript.com/learn/conditionals>