INDUSTRIAL ORIENTED MINI PROJECT

Report

On

PDFVOICE : AI-POWERED PDF INTO SPEECH CONVERTER USING MERN STACK

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

BACHELOR OF TECHNOLOGY

In

INFORMATION TECHNOLOGY

By

Pendyala Jaya Charan - 22261A1247

Chenemoni Vaishnavi- 22261A1214

Under the guidance of

Dr. N. Sree Divya

Assistant Professor, Department of IT

DEPARTMENT OF INFORMATION TECHNOLOGY

MAHATMA GANDHI INSTITUTE OF TECHNOLOGY (AUTONOMOUS)

(Affiliated to JNTUH, Hyderabad; Eight UG Programs Accredited by NBA; Accredited by NAAC with ‘A++’ Grade)

Gandipet, Hyderabad, Telangana, Chaitanya Bharati (P.O), Ranga Reddy District, Hyderabad– 500075, Telangana

2024-2025

CERTIFICATE

This is to certify that the **Industrial Oriented Mini Project** entitled **PDF VOICE : AI-POWERED PDF INTO SPEECH CONVERTER USING MERN STACK** submitted by **Pendyala Jaya Charan (22261A1247)** and **Chenemoni Vaishnavi (22261A1214)** in partial fulfillment of the requirements for the Award of the Degree of Bachelor of Technology in Information Technology as specialization is a record of the bona fide work carried out under the supervision of **Dr. N. Sree Divya**, and this has not been submitted to any other University or Institute for the award of any degree or diploma.

Internal Supervisor:	**IOMP Supervisor:**
Dr. N. Sree Divya	**Dr. U. Chaitanya**
Assistant Professor	Assistant Professor
Dept. of IT	Dept. of IT

EXTERNAL EXAMINAR	**Dr. D. Vijaya Lakshmi**
	Professor and HOD
	Dept. of IT

DECLARATION

We hear by declare that the **Industrial Oriented Mini Project** entitled **PDF VOICE : AI-POWERED PDF INTO SPEECH CONVERTER USING MERN STACK** is an original and bona fide work carried out by us as a part of fulfilment of Bachelor of Technology in Information Technology, Mahatma Gandhi Institute of Technology, Hyderabad, under the guidance of **Dr.N.Sree Divya** , Assistant Professor, Department of IT, MGIT.

No part of the project work is copied from books /journals/ internet and wherever the portion is taken, the same has been duly referred in the text. The report is based on the project work done entirely by us and not copied from any other source.

Pendyala Jaya Charan - 22261A1247
Chenemoni Vaishnavi - 22261A1214

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success. They have been a guiding light and source of inspiration towards the completion of the **Industrial Oriented Mini Project.**

We would like to express our sincere gratitude and indebtedness to our project guide **Dr. N. Sree Divya, Assistant Professor**, Dept. of IT, who has supported us throughout our project with immense patience and expertise.

We are also thankful to our honorable Principal of MGIT **Prof. G. Chandramohan Reddy** and **Dr. D. Vijaya Lakshmi**, Professor and HOD, Department of IT, for providing excellent infrastructure and a conducive atmosphere for completing this **Industrial Oriented Mini Project successfully.**

We are also extremely thankful to our Project IOMP supervisor **Dr. U. Chaitanya,** Assistant Professor, Department of IT, and senior faculty **Mrs.B.Meenakshi,** Assistant Professor Department of IT for their valuable suggestions and guidance throughout the course of this project.

We convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed.

Finally, we would like to take this opportunity to thank our families for their support all through the work. We sincerely acknowledge and thank all those who gave directly or indirectly their support for completion of this work.

Pendyala Jaya Charan - 22261A1247
Chenemoni Vaishnavi - 22261A1214

ABSTRACT

The PDF to Voice Converter is an intelligent web-based application designed to transform PDF documents into audio files, making digital content more accessible and user-friendly. Built using a MERN-stack frontend and a Python Flask backend, the system allows users to upload PDF files, select a preferred language, and receive an audio output in MP3 format. It supports multiple languages by translating the document's text using Google Translate API and converting it to speech using the gTTS (google Text-To-Speech) engine. The frontend offers a modern interface for file upload, language selection, audio playback, and download. This project not only aids individuals with visual impairments or reading difficulties but also promotes convenience for users who prefer to consume content in auditory form. With adjustable features and seamless integration, the system demonstrates practical utility in education, accessibility, and content consumption.

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

1. INTRODUCTION

1.1 MOTIVATION

In an increasingly digital and information-driven world, documents in PDF format have become a standard for sharing and archiving important information. However, access to this content is not always equitable. People with visual impairments, learning difficulties like dyslexia, or even those with tight schedules often face barriers in reading and understanding large amounts of text. This inspired the idea of converting textual documents into voice—a natural and intuitive medium of communication that can be consumed easily and conveniently.

With the rise of artificial intelligence and cloud-based services, converting text to speech has become more accurate, expressive, and multilingual. The motivation behind this project was to harness these technologies to create a tool that can automatically extract text from PDF files, translate it to a chosen language, and convert it into natural-sounding audio. This would not only improve accessibility but also serve as a productivity tool for students, professionals, and content consumers who prefer listening over reading, or who wish to multitask while learning or reviewing content.

Furthermore, the integration of language translation adds an extra layer of value, helping break down linguistic barriers and allowing content to reach a more diverse global audience. By giving users the ability to customize the language and listen to the content at their convenience, the project aims to empower users from all backgrounds to engage with digital content more effectively. Ultimately, this initiative is driven by a vision to make information universally accessible, inclusive, and easier to consume through the power of speech.

1.2 PROBLEM STATEMENT

In today’s information-rich digital landscape, reading documents—especially in PDF format—is a routine necessity across education, business, healthcare, and government sectors. However, while PDFs offer a universal standard for document sharing and preservation, they also pose significant accessibility challenges. A large segment of the population, including people with visual impairments, learning disabilities such as dyslexia, or elderly users with declining eyesight, often struggle to engage with and understand textual content in these files. In parallel, even fully-abled users increasingly seek more flexible, efficient ways to consume information, especially when multitasking or on the move. This raises a crucial question: how can we make digital text content more inclusive, interactive, and easier to consume?

Another significant challenge is the language barrier. Many important documents are written in languages that the reader may not understand fluently. Translation services exist, but they often involve copying and pasting content or relying on manual processes. This is not only inconvenient but also time-consuming and error-prone. Additionally, traditional screen readers or text-to-speech tools often lack user control, fail to provide downloadable output, or do not support multilingual content effectively. Users are left with rigid tools that don’t adapt to their personal preferences or needs.

Furthermore, with the rapid increase in mobile-first lifestyles and smart assistants, there's a growing shift towards voice-based content consumption. Podcasts, audiobooks, and voice notes are becoming mainstream, yet transforming static, written documents like PDFs into dynamic, listenable content remains underdeveloped in many applications. There’s a noticeable gap in tools that combine PDF processing, language translation, and real-time voice output in a single, user-friendly, and customizable platform.

This project aims to directly address these issues by developing an intelligent PDF to Voice Converter. The system empowers users to upload a PDF file, choose their preferred output language, and receive a high-quality audio file that reads the content aloud. It eliminates the need for manual reading, supports multilingual translation, and provides downloadable voice output that can be used anytime, anywhere. By making content both more accessible and more engaging, the project strives to bridge the gap between static digital documents and modern, voice-enabled user experiences.

1.3 EXISTING SYSTEM

Several existing systems are available that convert PDF documents into speech with features such as language selection and adjustable voice speed. One commonly used tool is Adobe Acrobat Reader, which has a built-in "Read Aloud" feature. However, it relies on system-installed voices and offers limited control over language selection and voice speed. Natural Reader is another popular tool that is available both online and as a desktop application. It supports multiple languages and voices, and allows users to adjust the speed of speech, but its free version offers limited access to high-quality voices. Balabolka is a free Windows application that supports various text-to-speech engines like SAPI5, giving users flexibility in language and voice settings, as well as full control over speed and pitch. However, it may require converting PDF to plain text for better results.

Google Text-to-Speech, mainly used on Android devices, supports dozens of languages and adjustable speed, but typically depends on third-party apps for PDF reading functionality. TTSReader is a web-based application that lets users upload text or files and listen to them in various voices and languages, although it may struggle with longer PDF documents in its free version. Additionally, Microsoft Edge includes a "Read Aloud" feature that works well with accessible PDFs and provides a range of Microsoft voices with adjustable speed settings. While these systems offer useful functionality, they often lack an integrated, user-friendly interface for simultaneously handling PDF input, language switching, and voice customization—an area where a custom project can provide enhanced usability and performance.

1.3.1 Limitations

- **Limited Language and Voice Options:**
  Many tools rely on system-installed or default voices, restricting the number of available languages and natural-sounding voice choices.
- **Poor Handling of Complex PDFs:**
  Some systems struggle to read scanned PDFs, tables, or multi-column layouts correctly, leading to inaccurate or incomplete speech output.
- **Lack of Integrated Controls:**
  Most tools do not offer seamless in-app controls for adjusting speech speed, switching languages, or selecting voices in real time, reducing user flexibility and customization.

1.4 PROPOSED SYSTEM

The proposed system is a Python-based application that converts the contents of a PDF file into speech, with the added features of language selection and voice speed adjustment. Unlike many existing tools that are limited by system voices or lack real-time controls, this system provides users with a simple and customizable interface to interact with. It extracts readable text from PDF documents and uses text-to-speech (TTS) technology to convert it into audio output.

The system supports multiple languages by integrating with TTS engines like Google Text-to-Speech (gTTS) or pyttsx3, allowing users to choose their preferred language for audio playback. Additionally, users can adjust the speaking speed to suit their listening preferences—whether they want slower speech for better understanding or faster speech for quicker consumption of content. The system is especially useful for visually impaired users, language learners, or anyone who prefers listening to content instead of reading.

1.4.1 ADVANTAGES

- **Multilingual Support:**
  Users can choose from a wide range of languages, making the system accessible to speakers of different languages.
- **Adjustable Voice Speed:**
  The system allows users to control the speed of the voice output, enhancing listening comfort and comprehension.
- **User-Friendly Interface:**
  Offers a simple and intuitive interface for selecting PDF files, languages, and speed options.
- **Accessibility for Visually Impaired:**
  Converts written content into speech, making PDFs accessible to users with visual impairments.
- **Offline Functionality (with pyttsx3):**
  When using offline engines like pyttsx3, the system can work without an internet connection.
- **Customizable Output:**
  Users can easily modify voice properties such as rate and volume to suit their preferences.
- **Improves Productivity:**
  Allows users to listen to documents while multitasking, saving time and effort.

1.5 OBJECTIVES

- Convert PDF text to speech.
- Allow users to select the language of the voice.
- Provide options to adjust the speech speed.
- Ensure the system is easy to use.
- Support both online and offline TTS engines.
- Enable playback control (play, pause, stop).
- Handle both simple and multi-page PDFs.
- Improve accessibility for visually impaired users.
- Maintain clarity and accuracy of the spoken output.
- Offer a lightweight and portable application.

1.6 HARDWARE AND SOFTWARE REQUIREMENTS

Software Requirements:

Software:

The system is developed using Visual Studio Code (VS Code) as the integrated development environment (IDE) for Python and JavaScript. VS Code offers a flexible and efficient environment for both backend (Python) and frontend (React) development, supporting extensions for debugging, formatting, and live previews.

• **Primary Language:**

The core logic of the application is developed in Python, which handles PDF file reading, text extraction, and integration with text-to-speech (TTS) engines. Libraries like PyPDF2, gTTS, and pyttsx3 are used to support both online and offline voice generation.

• **Frontend Framework:**

The user interface is built using React, a popular JavaScript library for building modern and responsive web applications. React provides a dynamic and interactive frontend, allowing users to upload PDFs, choose languages, adjust voice speed, and control playback through a smooth and intuitive UI.

- **Backend Framework:**

The backend logic is written in Python, managing tasks such as extracting text from PDFs, converting text to speech, and returning audio output to the frontend. The Python backend can run via lightweight servers such as Flask or FastAPI, depending on deployment needs.

- **Database:**

The system does not require a complex database but can optionally use SQLite3 or JSON files to store user preferences, past usage history, or selected settings for enhanced user experience.

- **Frontend Technologies:**

React JS: For building a modular and interactive user interface.

HTML5: For basic structure and layout.

CSS3: For styling and visual design.

JavaScript (ES6): For adding interactivity and handling frontend logic.

Bootstrap 4 / Tailwind : For responsive design across devices and better UI components.

1.6.1 HARDWARE REQUIREMENTS

Operating System:

The system is designed to run on Windows, macOS, and Linux operating systems, ensuring broad compatibility with the development environment and user machines.

Processor:

A processor with at least an Intel Core i3 or equivalent is recommended to smoothly perform PDF text extraction and text-to-speech processing. For faster performance, an Intel i5 or higher is preferred.

RAM:

A minimum of 4GB RAM is required to efficiently handle PDF processing and voice synthesis tasks. However, 8GB or more is recommended for smoother multitasking and handling larger PDF files.

Storage:

At least 500 MB of free disk space is needed for installing the software dependencies, libraries, and temporary audio file storage during speech synthesis.

2. LITERATURE SURVEY

Wang et al. (2023) proposed a multilingual Transformer-based OCR and TTS pipeline that effectively handles text extraction and voice synthesis in multiple languages. This approach significantly enhances contextual accuracy, enabling users to interact with documents containing diverse languages seamlessly. However, the system requires a large memory footprint, limiting its feasibility on resource-constrained devices. Moreover, it lacks optimization for deployment on embedded systems, which is essential for integrating such models into web-based solutions like MERN stack frameworks. The study suggests that future work should focus on lightweight model compression and efficient inference strategies to broaden its applicability.[1].

Tan & Ibrahim (2023) explored a lightweight OCR solution using Vision Transformers (ViT), targeting efficient text recognition on edge devices and mobile platforms. The system achieves high efficiency and low computational overhead, making it well-suited for integration into web applications built on the MERN stack. However, this method experiences a slight drop in recognition accuracy compared to larger, more complex models, particularly with stylized fonts and complex document layouts. The authors highlight the need for improved font handling and layout-aware OCR models that can generalize better across diverse PDF content, a challenge that remains relevant for real-world deployments.[2].

Nguyen et al. (2023) investigated a real-time TTS system using diffusion models to produce natural and expressive voice output from textual data. The model demonstrates superior performance in generating human-like speech, making it ideal for dynamic voice rendering in web-based platforms. However, training these models requires substantial computational resources, posing a challenge for real-time mobile and embedded applications. Additionally, the study points out that faster inference is crucial to meet the needs of interactive web systems like those built on the MERN stack, which demand low-latency audio synthesis. This research gap opens the door for developing optimized TTS pipelines that balance quality and efficiency.[3].

Ahmed et al. (2023) presented a context-aware PDF reader with AI-based voice personalization, focusing on delivering personalized and dynamic audio experiences for end users. This approach leverages user context and preferences to tailor voice output, enhancing engagement and accessibility. Despite its merits, the system suffers from high latency on older devices and requires significant optimization to achieve real-time performance. The study also emphasizes the importance of seamless integration with web frameworks like the MERN stack, which can enable personalized TTS delivery through interactive user interfaces. Future research should prioritize low-latency personalization techniques and efficient deployment strategies to overcome these limitations.[4].

Table 2.1 Literature Survey of Pdf to voice converter using MERN stack

Ref	Author & year of publications	Journal / Conference	Method / Approach	Merits	Limitations	Research Gap
[1]	Wang et al., (2023)	IEEE Access	Multilingual Transformer-based OCR and TTS pipeline	Supports multiple languages; improved contextual accuracy	Requires large memory footprint	Lacks optimization for embedded systems
[2]	Tan & Ibrahim (2023)	Elsevier – Pattern Recognition Letters	Lightweight OCR with Vision Transformers (ViT)	High efficiency on edge devices	Slight drop in accuracy compared to larger models	Requires better handling of stylized fonts in PDFs
[3]	Nguyen et al., (2023)	ACM Multimedia	Real-time TTS using diffusion models	Produces natural and expressive speech	Training requires significant compute	Needs faster inference for real- time mobile use
[4]	Ahmed et al., (2023)	Wiley - Expert Systems	Context-aware PDF reader with AI- based voice personalization	Personalized and dynamic audio output	High latency on older devices	Optimization needed for real-time personalization

3. ANALYSIS AND DESIGN

The development of the PDF to Voice Converter system begins with a thorough analysis of user requirements, system capabilities, and technical constraints. The primary focus is to create a seamless experience where users can upload PDF documents, select their preferred language for voice output, and adjust the speech speed as per their listening comfort. This necessitates a design that efficiently extracts text from various PDF formats while maintaining the integrity of the content and supporting multiple languages and voice options.

The system architecture is designed to separate concerns between the frontend user interface and the backend processing engine. The frontend, built with **React**, offers an interactive platform for users to interact with the application, providing options to upload files, select languages, and control playback speed dynamically. Meanwhile, the backend, developed in Python, handles the core functionalities including PDF text extraction, language processing, and text-to-speech conversion using engines like **pyttsx3** and **gTTS.** This modular design ensures scalability and easy maintenance.

To achieve accurate and high-quality speech output, the system incorporates a flexible text-to-speech pipeline that supports both online and offline voice synthesis methods. The backend design also includes error handling and preprocessing steps to manage diverse PDF structures and formats, ensuring consistent text extraction. Additionally, the design considers responsiveness and user accessibility, allowing real-time adjustments to voice speed and language selection without interrupting the audio playback. This comprehensive approach to analysis and design ensures that the system meets user expectations for functionality, usability, and performance.

3.1 MODULES

Backend

- **Flask:** Used to create the web server and handle incoming API requests (/convert).
- **flask_cors:** Enables Cross-Origin Resource Sharing, so your React frontend can talk to the Flask backend.
- **PyPDF2**: Extracts plain text from uploaded PDF files.
- **gtts** (Google Text-to-Speech): Converts the translated text into an MP3 audio file.
- **googletrans** (version 4.0.0-rc1): Automatically translates the text into the selected language before converting it to speech.
- **Io**: Helps create in-memory binary streams to hold the MP3 data for sending in the response.
- **Tempfile-** Creates a temporary MP3 file on the server before sending it to the client.

Frontend

- **React**: Core library used to build the user interface, including components like upload, dropdown, and audio list.
- **Axios:** Used to send HTTP POST requests to the backend, carrying the PDF file and selected language.
- **react-router-dom**: Manages page routing, allowing navigation between different components (/ and /pdftovoice).
- **Bootstrap:** Used for styling and layout of the UI — buttons, forms, cards, etc.
- **HTML5 <audio> tag:** Enables audio playback directly in the browser once the MP3 is returned from the backend.

3.2 ARCHITECTURE

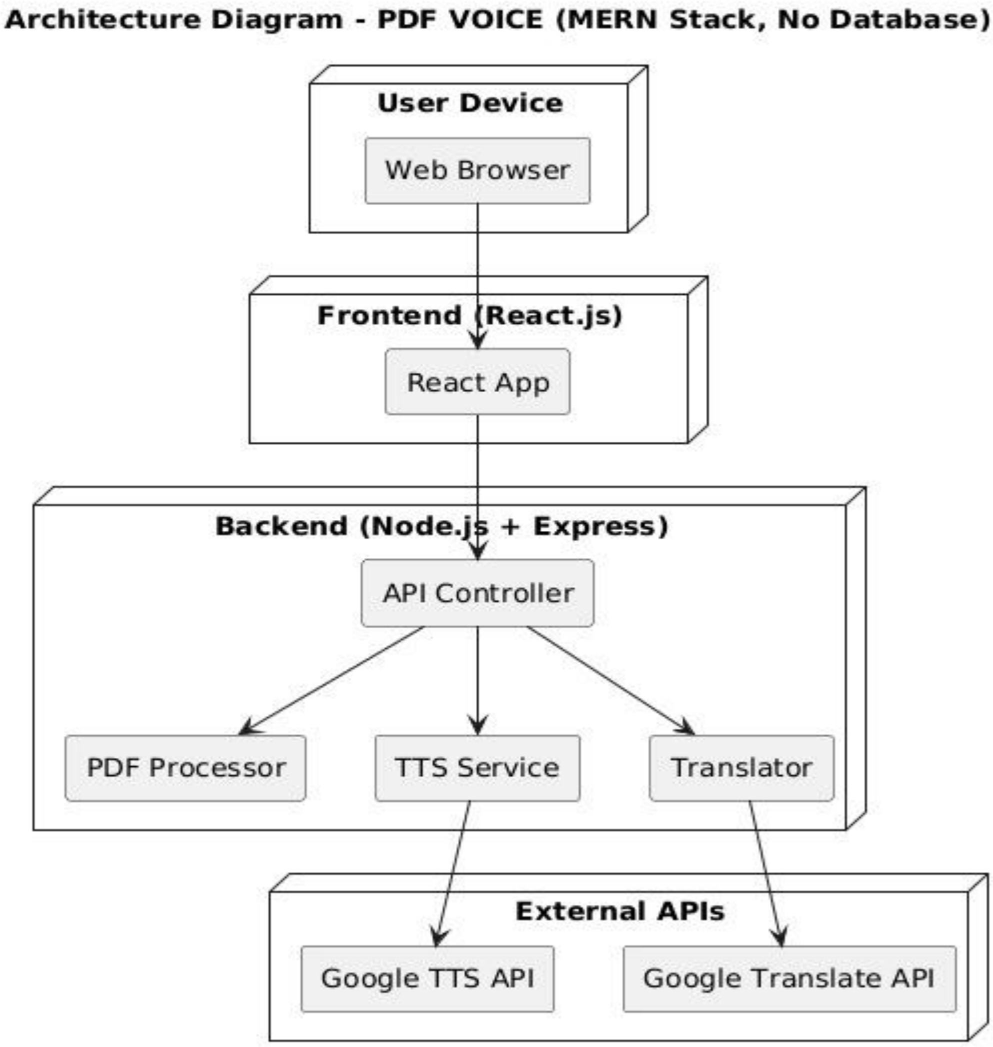


Fig. 3.2.1 Architecture of Pdf to voice converter using MERN stack

The architecture of the "PDF to Voice Converter" project is designed using a modular, client-server based web application model, ensuring scalability, maintainability, and ease of interaction between the user and the system. The system is divided into two main segments: the frontend (client) and the backend (server), each handling specific responsibilities while working seamlessly together. The frontend is developed using React.js, a robust JavaScript library that manages the user interface and interaction. It allows users to upload a PDF file, select a preferred output language, and initiate the conversion process through an intuitive and responsive graphical interface. Upon submitting the input, the React application sends a POST request to the Flask backend using Axios, a promise-based HTTP client.

The backend, powered by Python’s Flask framework, serves as the processing unit of the application. It receives the uploaded PDF and the selected language, then performs a series of operations to convert the document into speech. The PDF content is first parsed using the PyPDF2 library to extract raw text from each page. The extracted text is then passed through the googletrans library, which translates the content into the language specified by the user.

The entire flow is tightly coupled through clear API interaction, and the architecture ensures asynchronous, non-blocking communication, offering a smooth user experience. The use of in-memory data streams (via io.BytesIO) prevents unnecessary file I/O on the server and enhances performance. This modular structure also allows future enhancements such as voice customization, additional language support, or even integration with cloud-based translation and TTS services. The architecture supports robust, secure, and real-time functionality for transforming static documents into dynamic, accessible audio formats, making it a highly practical solution for visually impaired users, language learners, and multitasking professionals alike.

3.3 UML DIAGRAMS

3.3.1 USE CASE DIAGRAMS

A use case diagram is a visual representation that depicts the interactions between various actors and a system, capturing the ways in which users or external entities interact with the system to achieve specific goals. It is an essential tool in system analysis and design, often used in software engineering and business analysis. In a use case diagram, actors are entities external to the system that interact with it, and use cases are specific functionalities or features provided by the system as seen in Fig. 3.3.1.1. These interactions are represented by lines connecting actors to use cases. The diagram helps to illustrate the scope and functionality of a system, providing a high-level view of how users or external entities will interact with it.

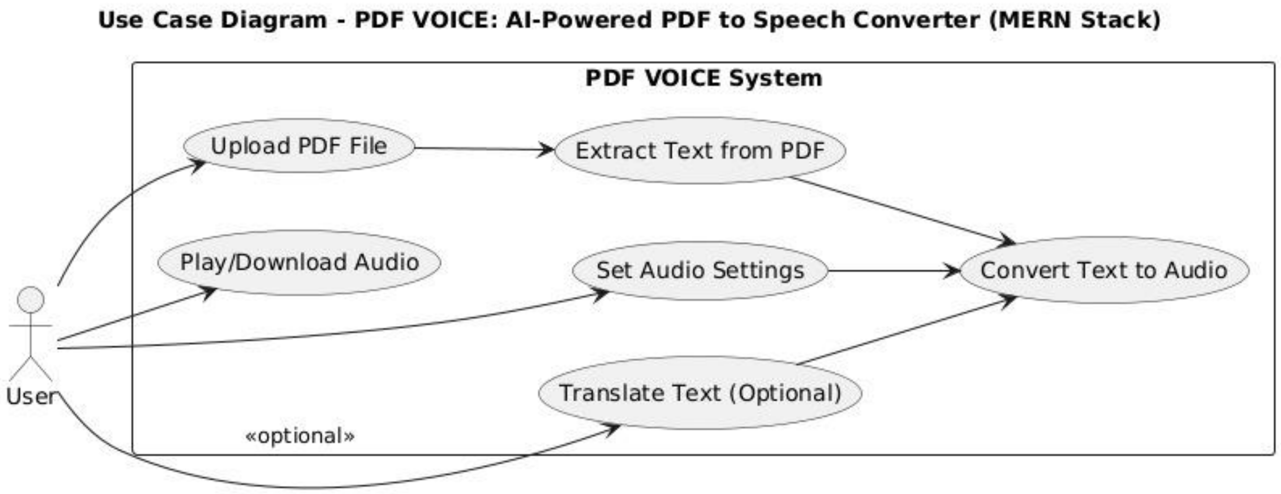


Fig. 3.3.1.1 Use Case Diagram

Actors:

1. **User:** The user interacts with the system by uploading a PDF file and selecting the desired output language. They initiate the conversion process and receive an audio file in return, which can be played or downloaded directly from the interface.
2. **System:** The system comprises a Flask-based backend that processes the uploaded PDF using PyPDF2, translates the extracted text using googletrans, and converts it into an MP3 file using gTTS. The audio file is then streamed back to the user through the frontend for immediate access.

Use Cases:

- **Upload PDF Documennt**
  The user selects and uploads a PDF file from their local device using the interface provided on the frontend. This file serves as the primary input for text extraction and conversion.
- **Select Output Language**
  The user chooses a preferred language from a dropdown menu. This determines the target language in which the extracted text will be translated before being converted to speech.
- **Convert Text to Speech**
  Upon clicking the convert button, the system processes the PDF, extracts the text, translates it, and uses a text-to-speech engine to generate an MP3 audio file.
- **Play Generated Audio**
  After successful conversion, the audio file is embedded in the frontend using an HTML5 <audio> player. The user can listen to the spoken version of the PDF content.

- **Download Audio File**
  Users are provided with an option to download the generated MP3 file. This enables offline access to the converted audio for future use.
- **View Conversion History (Client-side)**
  On the frontend, a list of previously converted files in the current session is maintained, showing filename, time of conversion, and download/play options.

3.3.2 CLASS DIAGRAM

A class diagram visually represents the static structure of the PDF to Voice Converter system, detailing the main classes, their attributes, methods (operations), and the relationships among them. It serves as an essential tool in object-oriented design and software engineering by providing a blueprint of the system's architecture.

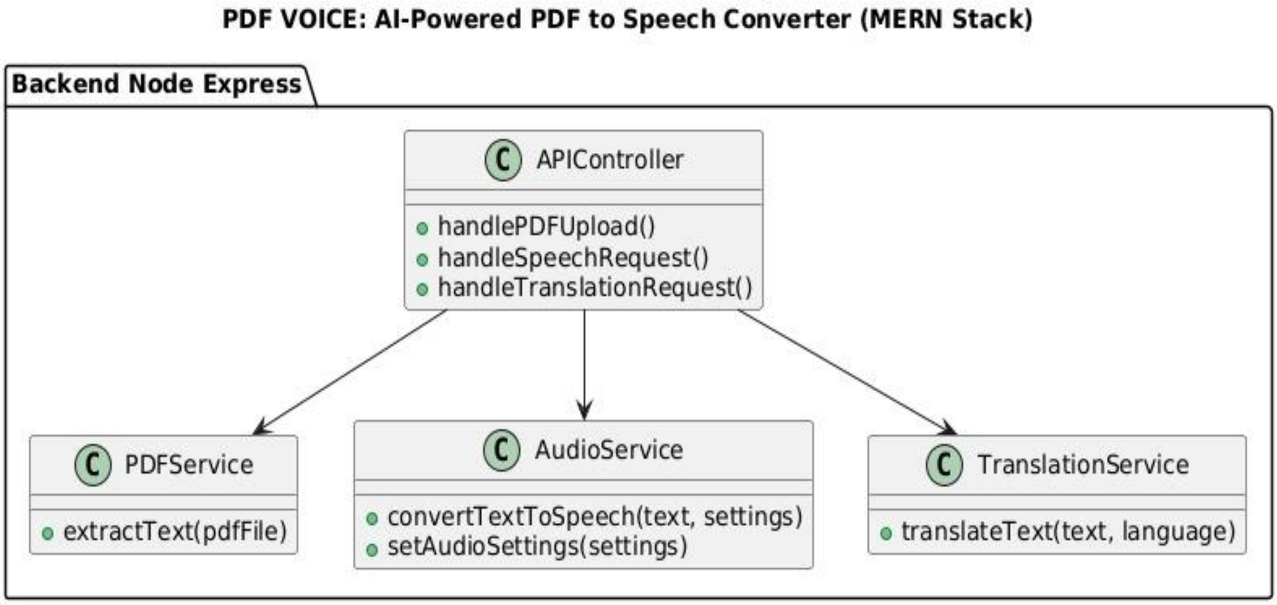

PDF VOICE: AI-Powered PDF to Speech Converter (MERN Stack)
Backend Node Express
C APIController
handlePDFUpload()
handleSpeechRequest()
handleTranslationRequest()
C PDFService
extractText(pdfFile)
C AudioService
convertTextToSpeech(text, settings)
setAudioSettings(settings)
C TranslationService
translateText(text, language)

Fig. 3.3.2.1 Class Diagram

User → Backend Server: The User interacts with Backend Server to upload PDF files and select voice settings (such as language & speed). The Backend Server validates user requests & processes PDF files.

Backend Server → Database: The Backend Server stores user data, preferences, and converted audio file metadata in the Database. It retrieves stored user settings to personalize the audio output.

Backend Server → PDF Processor: The Backend Server sends the uploaded PDF to the PDF Processor module. The PDF Processor extracts textual content from the PDF, including text formatting and structure.

Backend Server → Text-to-Speech (TTS) Engine: The extracted text from the PDF is passed to the TTS Engine. The TTS Engine converts the textual content into audio data using selected voice parameters.

TTS Engine → Audio Output: The generated audio data is formatted into playable audio files (e.g., MP3 or WAV). These audio files are sent back to the Backend Server for storage or direct download.

Backend Server → Frontend Client: The Backend Server sends the audio output and conversion status back to the Frontend Client. The Frontend Client presents the audio player and options for downloading or streaming the converted speech.

System Flow:

User Interaction: The user uploads a PDF file through the Frontend interface. The user selects preferred voice customization options like language, gender, and speech rate.

Text Extraction: The Backend Server receives the PDF and forwards it to the PDF Processor. The PDF Processor extracts clean and structured text from the PDF document.

Text-to-Speech Conversion: The Backend Server sends the extracted text to the TTS Engine. The TTS Engine synthesizes speech audio from the text, applying the user's selected voice parameters.

Audio Delivery: The synthesized audio is returned to the Backend Server and stored temporarily or permanently in the Database. The Frontend receives the audio file URL or stream and allows the user to listen or download the audio.

Customization and Feedback: The user can adjust voice settings and convert the PDF again for different audio outputs. User preferences and conversion history are managed in the Database fo

a personalized experience.

3.3.3 ACTIVITY DIAGRAM FOR RHYTHM RESTORE

An Activity Diagram is a type of behavioral diagram used in Unified Modeling Language (UML) to represent the flow of control or data through the system as seen in Fig. 3.3.3.1. It focuses on the flow of activities and actions, capturing the sequence of steps in a particular process or workflow. Activity diagrams are commonly used to model business processes, workflows, or any sequential activities in a system.

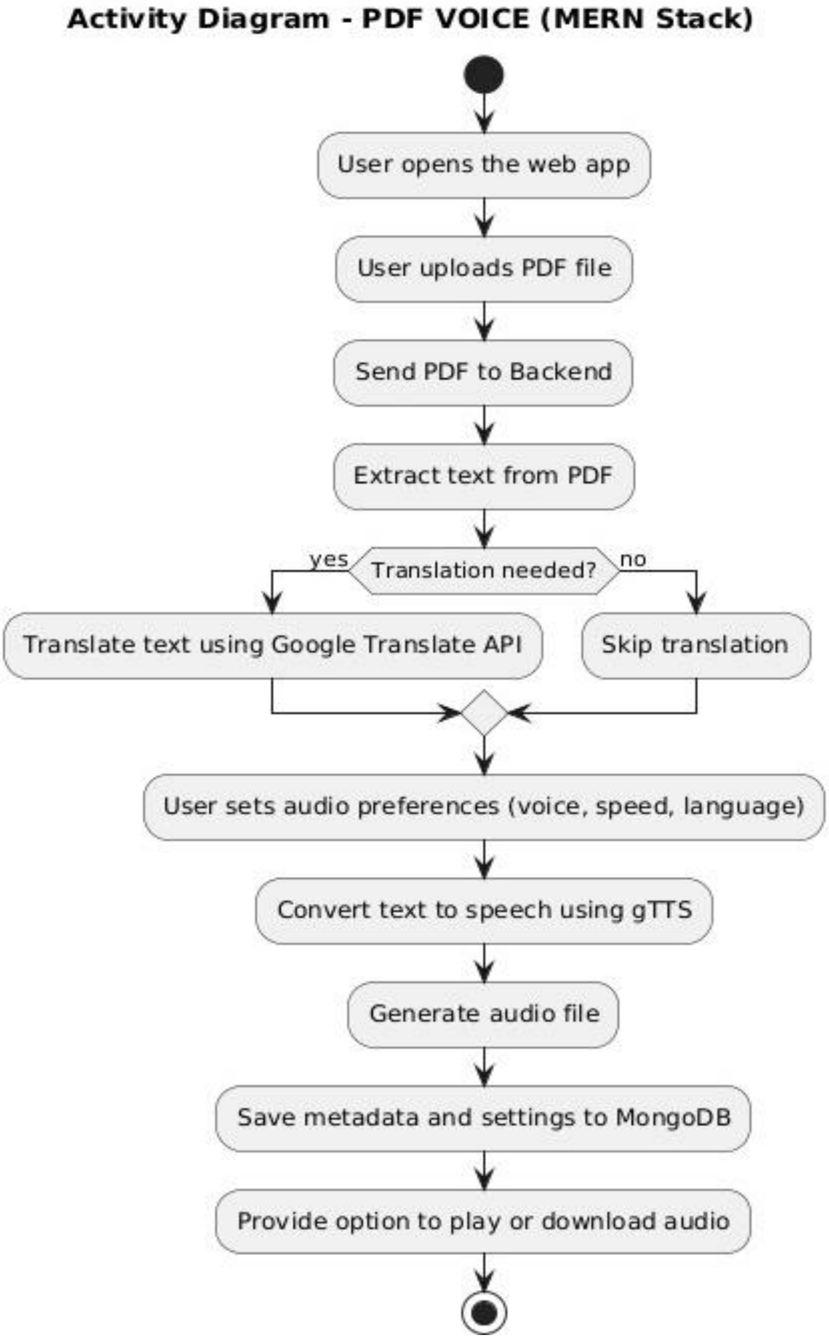

Activity Diagram - PDF VOICE (MERN Stack)
User opens the web app
User uploads PDF file
Send PDF to Backend
Extract text from PDF
Translation needed?
yes
no
Translate text using Google Translate API
Skip translation
User sets audio preferences (voice, speed, language)
Convert text to speech using gTTS
Generate audio file
Save metadata and settings to MongoDB
Provide option to play or download audio

Fig. 3.3.3.1 Activity Diagram

Flow Explanation:

Open Application:

The user opens the PDF to Voice Converter web application.

Upload PDF:

The user uploads a PDF document through the frontend interface.

File Validation:

The system verifies whether the uploaded file is a valid PDF. If invalid, the user is prompted to re-upload.

Select Voice Settings:

The user selects voice preferences such as language, gender, and speech speed.

Extract Text from PDF:

The backend processes the PDF file to extract textual content.

Convert Text to Speech:

The extracted text and selected voice settings are passed to the Text-to-Speech (TTS) engine, which synthesizes the audio.

Generate Audio File:

The synthesized speech is converted into an audio file format (e.g., MP3).

Provide Audio Output:

The system provides the audio file to the user for playback or download.

End of Workflow:

The user may choose to convert another PDF or close the application.

3.3.4 SEQUENCE DIAGRAM

A sequence diagram illustrates the flow of interactions between actors and system components over time as seen in Fig. 3.3.4.1, emphasizing the order in which messages are exchanged to achieve specific functionalities. Actors represent external entities that interact with the system, while lifelines depict the system components involved in the process. Messages are shown as arrows, indicating the flow of information or actions between these elements. By providing a step-by-step view of workflows, sequence diagrams help in understanding and designing the dynamic behavior of a system.

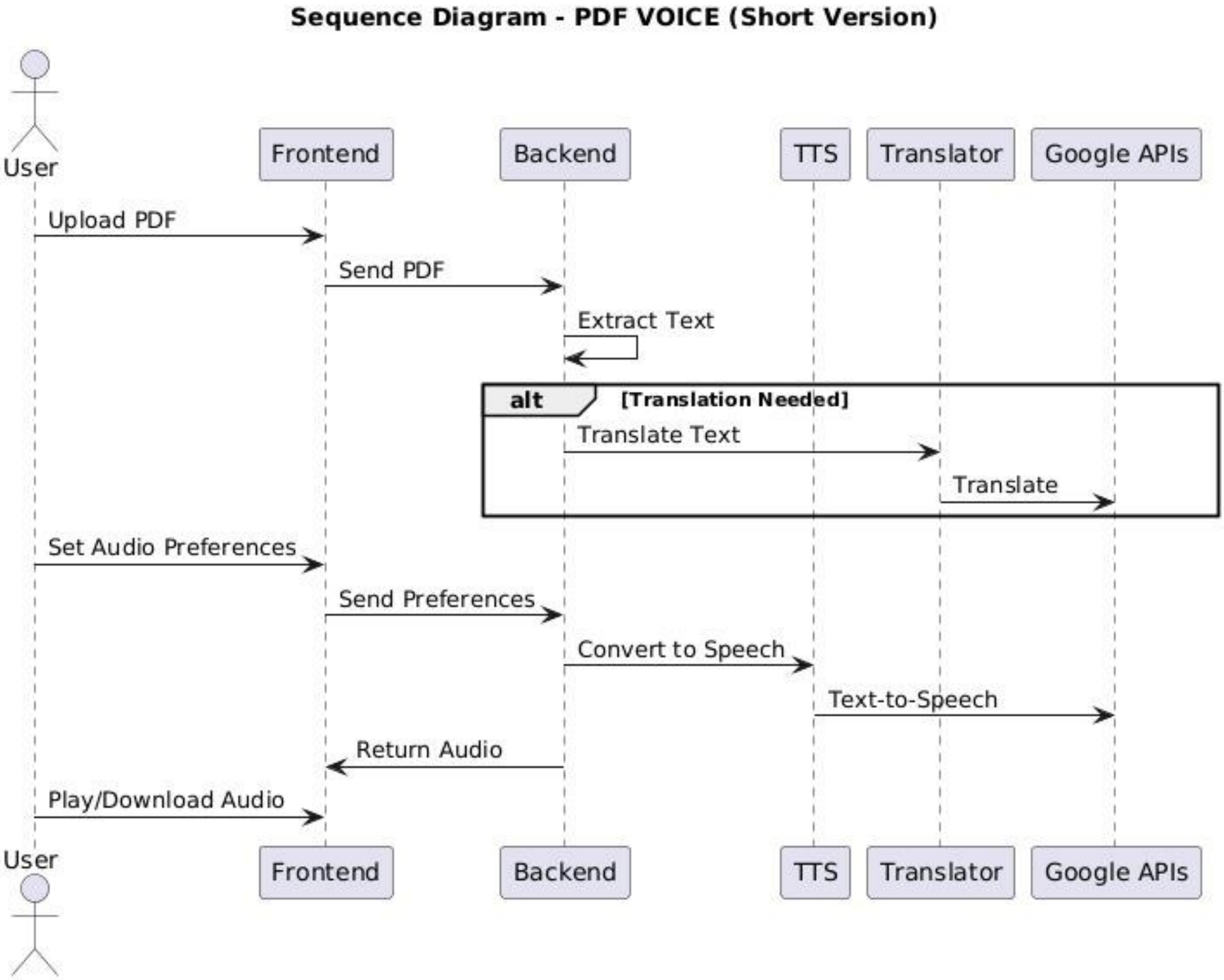


Fig. 3.3.4.1 Sequence Diagram

Key Interactions and Relationships

- **User and System:** Upload and Conversion Request
  The user uploads a PDF document and requests a voice conversion through the frontend.

- **System and Database:** User Preferences (Optional)
  The system may store user preferences (such as voice type, language, or speech speed) in the database for personalized settings in future sessions.

- **System and Text Extraction Module:** Extracting Text
  The backend processes the uploaded PDF using a text extraction library or service to retrieve textual content from the document.

- **System and Text-to-Speech Engine:** Speech Synthesis
  The extracted text is passed to the Text-to-Speech engine, which converts it into an audio file (e.g., MP3 format).

- **System and User:** Returning Audio Output
  The system provides the generated audio file back to the user through the frontend interface, allowing them to listen or download the speech-converted version of the PDF.

3.3.5 COMPONENT DIAGRAM

A Component Diagram is a type of structural diagram used in software engineering to represent the components of a system and how they interact or depend on each other. It shows how the components (which could be software modules, subsystems, or other significant parts) are organized and connected within a system. In this diagram, each component encapsulates a set of related functionalities and interfaces as shown in Fig.3.3.5.

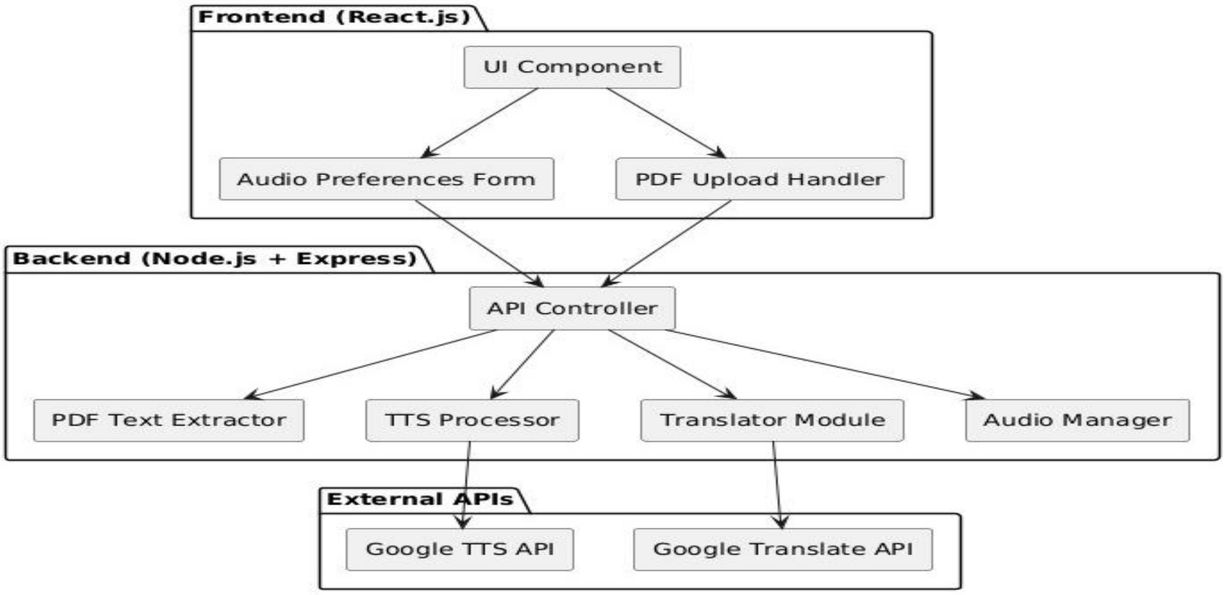


Fig. 3.3.5.1 Component Diagram

Main Components:

User Interface:

The frontend component, built using React.js, where users upload PDF files and request audio conversion. It allows users to interact with the system, view upload status, and listen to or download the converted audio file.

- **Express.js Server:**

  The backend server built using Express.js (running on Node.js) that handles requests from the User Interface. It manages file uploads, text extraction, communication with the Text-to-Speech engine, and response handling.

- **Text Extraction Module:**

  A backend module that processes uploaded PDF files to extract their text content using libraries like pdf-parse or pdf-lib.

- T**ext-to-Speech (TTS) Engine:**

  The module that converts the extracted text into speech. It could use third-party services (like Google TTS, Amazon Polly) or Node.js libraries (like say.js) to generate audio files (e.g., MP3 format).

- **Database:**

  MongoDB is used to store user data, upload history, audio file metadata, and optional user preferences (such as voice type or speed) to enhance user experience and personalization.

- **Audio File Storage:**
  A storage component (such as the file system, cloud storage, or MongoDB GridFS) that saves the generated audio files, making them available for user playback and download.
- **Progress Module (Optional):**
  If the system includes progress tracking (e.g., how many files converted, duration of usage), this module would store and update user progress and history.
- **Admin Dashboard (Optional):**
  An administrative interface that allows admins to manage users, view system usage, monitor conversion requests, and oversee the overall health of the system.

3.4 METHODOLOGY

Data Acquisition

PDF Input: The system accepts PDF files uploaded by users through the frontend interface. These files may contain text in various languages and formats, which need to be processed accurately.

Text Extraction: Using Python libraries such as PyPDF2 or pdfplumber, the system extracts text content from the uploaded PDF files. This step is crucial for converting the static document into a machine-readable format suitable for speech synthesis.

Voice Generation: The extracted text is passed to text-to-speech (TTS) engines like pyttsx3 (offline) or gTTS (online) for converting text into audible speech. The system supports multiple languages and allows customization of speech speed based on user input.

Model Steps:

- **Text Extraction**: The system extracts text from the uploaded PDF documents using Python libraries like PyPDF2 or pdfplumber. This step ensures that all readable content is accurately captured for further processing.
- **Language Detection and Selection:** Based on user input, the system identifies the target language for speech synthesis. It supports multiple languages by integrating with TTS engines that provide multilingual voice options.

- **Preprocessing:** The extracted text is cleaned and formatted to remove unnecessary characters, handle line breaks, and structure content for natural speech flow. This improves the clarity and quality of the voice output.
- **Speech Synthesis:** The preprocessed text is converted into audio using text-to-speech engines such as pyttsx3 for offline use or gTTS for online voice generation. Users can choose between engines depending on their preferences and connectivity.
- **Speed Adjustment:** The system allows users to adjust the voice playback speed, enhancing the listening experience by accommodate different preferences for speech rate.
- **Playback Control**: Audio playback is managed with features such as play, pause, stop, and restart, giving users full control over how they consume the spoken content.
- **Error Handling:** The system detects and manages errors such as unsupported PDF formats or missing text during online TTS processing, ensuring smooth operation.
- **Output Delivery:** The synthesized audio is streamed or saved as an audio file (e.g., MP3) for the user to listen to directly or download for offline use.

Models Used

In this project, the term "models" refers to the logical components or processing modules that work together to perform the core function of converting PDF documents into audio files. Although the project does not involve traditional machine learning models, it relies heavily on pre-trained and intelligent processing modules such as language translators and text-to-speech engines. Each of these models contributes a specific role in the overall workflow, forming a pipeline that transforms static textual input into dynamic audio output. The modular design also enhances the maintainability and scalability of the system.

- **Text Extraction Model – PyPDF2**
  Extracts text from each page of the uploaded PDF.
  Used inside the extract_text_from_pdf() function in server.py.
  Ensures only readable, valid content is passed on for translation.
- **Translation Model – googletrans**
  Translates the extracted text into the user-selected language.
  Triggered after text extraction, before speech synthesis.
  Makes the application multilingual and user-friendly.

- **Text-to-Speech (TTS) Model – gTTS**

  Converts the translated text into an MP3 audio file.

  Utilizes Google’s text-to-speech engine.

  Produces natural-sounding speech in multiple languages.

- **Temporary Storage Model – tempfile & io**

  tempfile.NamedTemporaryFile() is used to store the generated MP3 file.

  io.BytesIO() helps return the audio file as a stream in the response

- **File Upload & API Handling – Flask**

  Handles HTTP POST requests through the /convert endpoint.

  Accepts uploaded PDF files and language input.

  Sends audio as an HTTP response using send_file().

- **Frontend Integration Model – React + Axios**

  React provides the user interface for uploading PDFs and selecting language.

  Axios handles communication between frontend and Flask backend.

  Uses HTML5 <audio> tag for audio playback and dynamic file downloads.

4. CODE AND IMPLEMENTATION

4.1 CODE

Server.py

```
from flask import Flask, request, send_file, jsonify
from flask_cors import CORS
import PyPDF2
from gtts import gTTS
import io
import tempfile
from googletrans import Translator

app = Flask(__name__)
CORS(app)

# Function to extract text from PDF
def extract_text_from_pdf(pdf_file):
    pdf_reader = PyPDF2.PdfReader(pdf_file)
    text = ""
    for page in pdf_reader.pages:
        extracted = page.extract_text()
        if extracted:
            text += extracted
    return text

@app.route('/convert', methods=['POST'])
def convert_pdf_to_audio():
    pdf_file = request.files.get('file')
    language = request.form.get('language', 'en')

    if not pdf_file:
        return jsonify({"error": "No PDF file uploaded"}), 400

    text = extract_text_from_pdf(pdf_file)

    if not text:
        return jsonify({"error": "Could not extract text from the PDF"}), 400

    try:
        # Translate text to target language
        translator = Translator()
        translated = translator.translate(text, dest=language)
        translated_text = translated.text

        # Convert translated text to speech
        tts = gTTS(translated_text, lang=language)

        with tempfile.NamedTemporaryFile(delete=False, suffix=".mp3") as tmp:
            tts.save(tmp.name)
            tmp_path = tmp.name

        with open(tmp_path, "rb") as f:
            audio_data = io.BytesIO(f.read())

        audio_data.seek(0)

    except Exception as e:
        return jsonify({"error": f"Text-to-speech failed: {str(e)}"}), 500

    return send_file(audio_data, mimetype='audio/mp3', as_attachment=True,
download_name='output.mp3')

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

App.js

```
import Landing from './Landing';

import PdfToSpeech from './PDFtoVoice';

import {BrowserRouter as Router, Route,Routes} from 'react-router-dom'

export default function App(){

  return(

    <Router>

      <Routes>

        <Route path='/' element={<Landing/>}/>

        <Route path='/pdftovoice' element={<PdfToSpeech/>}/>

    </Routes>

    </Router>

  )

}
```

Pdftovoice.js:

```
import axios from "axios";
import { useState } from "react";

export default function PdfToSpeech() {
  const [file, setFile] = useState(null);
  const [language, setLanguage] = useState("en");
  const [loading, setLoading] = useState(false);
  const [audioList, setAudioList] = useState([]);

  const handleUpload = async () => {
    if (!file) return alert("Please upload a PDF file.");

    setLoading(true);

    const formData = new FormData();
    formData.append("file", file);
    formData.append("language", language);

    try {
      console.log("debug1");
      const response = await axios.post("http://127.0.0.1:5000/convert", formData, {
        headers: { "Content-Type": "multipart/form-data" },
        responseType: "blob",
      });
      console.log("debug2");
      if (response.status === 200) {
        const url = URL.createObjectURL(response.data);
        const filename = file.name.replace(".pdf", "") + "_" + Date.now();
        setAudioList((prevList) => [...prevList, { url, filename,time:new Date().toLocaleString() }]);
      } else {
        alert("Error processing the PDF.");
      }
    } catch (error) {
      console.error("Error:", error);
      alert("Something went wrong. Try again.");
    } finally {
      setLoading(false);
    }
  };

 const handleDownload = (audio) => {
  const link = document.createElement("a");
  link.href = audio.url;
  link.download = `${audio.filename}.mp3`;
  document.body.appendChild(link);
  link.click();
  document.body.removeChild(link);
};

  return (
    <div className="container-fluid py-5 bg-dark text-light min-vh-100">
      <div className="row justify-content-center">
        <div className="col-md-10 col-lg-8">

          <div className="text-center mb-5">
            <h1 className="display-4 fw-bold">PDF to Speech</h1>
            <p className="lead opacity-75">Transform your documents into audio with one click</p>
          </div>

          <div className="card border-0 shadow-lg rounded-4 overflow-hidden">
            <div className="card-header text-white p-4 border-0"
style={{backgroundColor:'#ff6b6b'}}>
              <h3 className="m-0 fw-bold">Upload Your Document</h3>
            </div>

            <div className="card-body p-4">
              <div className="row">
                <div className="col-md-8 mb-3 mb-md-0">
                  <div className="form-floating mb-3">
                    <input
                      type="file"
                      className="form-control form-control-lg"
                      id="pdfFile"
                      accept="application/pdf"
                      onChange={(e) => setFile(e.target.files[0])}
                    />
                    <label htmlFor="pdfFile" className="text-muted">Choose PDF file</label>
                  </div>
                </div>

                <div className="col-md-4">
                  <div className="form-floating">
                    <select
                      className="form-select form-select-lg"
                      id="languageSelect"
                      value={language}
                      onChange={(e) => setLanguage(e.target.value)}
                    >
                      <option value="en">English</option>
                      <option value="fr">French</option>
                      <option value="es">Spanish</option>
                      <option value="de">German</option>
                      <option value="it">Italian</option>
                    </select>
                    <label htmlFor="languageSelect" className="text-muted">Language</label>
                  </div>
                </div>
              </div>

              <div className="d-grid mt-4">
                <button
                style={{backgroundColor:'#ff6b6b',color:'white'}}
                  className={`btn btn-lg py-3`}
                  onClick={handleUpload}
                  disabled={loading || !file}
                >
                  {loading ? (
                    <>
                      <span className="spinner-border spinner-border-sm me-2" role="status" aria-
hidden="true"></span>
                      Processing...
                    </>
                  ) : (
                    <>Convert to Audio</>
                  )}
                </button>
              </div>

              {!file && !loading && (
                <div className="text-center mt-3 text-muted">
                  <small>Supported format: PDF</small>
                </div>
              )}

              {file && !loading && (
                <div className="alert alert-success mt-3 d-flex align-items-center">
                  <div className="me-3">
                    <i className="bi bi-file-earmark-text h4 mb-0"></i>
                  </div>
                  <div>
                    <strong>Ready to convert:</strong> {file.name}
                  </div>
                </div>
              )}
            </div>
          </div>

          {audioList.length > 0 && (
            <div className="mt-5">
              <h4 className="mb-4 border-bottom pb-2">Your Audio Files</h4>

              {audioList.map((audio, index) => (
                <div className="card mb-3 border-0 shadow-sm bg-white bg-opacity-10 text-light"
key={index}>
                  <div className="card-body p-3">
                    <div className="row align-items-center">
                      <div className="col-md-5">
                        <h6 className="text-truncate mb-0">
                          {audio.filename.split('_')[0]}
                        </h6>
                        <small style={{color:'white'}}>{audio.time}</small>
                      </div>
```

```
                      <div className="col-md-5">
                        <div className="rounded-3 bg-dark bg-opacity-50 p-2">
                          <audio controls className="w-100">
                            <source src={audio.url} type="audio/wav" />
                            Your browser does not support the audio element.
                          </audio>
                        </div>
                      </div>

                      <div className="col-md-2 mt-2 mt-md-0 text-md-end">
                        <button
                          className="btn btn-outline-light"
                          onClick={() => handleDownload(audio)}
                        >
                          <i className="bi bi-download me-1"></i> Save
                        </button>
                      </div>
                    </div>
                  </div>
                </div>
              ))}
            </div>
          )}
        </div>
      </div>
    </div>
  );
}
```

App.css:

```
body {
  margin: 0;
  padding: 0;
  font-family: 'Segoe UI', sans-serif;
  background: linear-gradient(-45deg, #1e3c72, #2a5298, #0f2027, #203a43);
  background-size: 400% 400%;
  animation: gradientBG 15s ease infinite;
  min-height: 100vh;
}

/* Animation keyframes */
@keyframes gradientBG {
  0% {
    background-position: 0% 50%;
  }
  50% {
    background-position: 100% 50%;
  }
  100% {
    background-position: 0% 50%;
  }
}

.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}
```

```
.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

Index.css

```
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
```

```
    <div id="stars-container" style="position: absolute; top: 20px; left: 20px; width: 160px; height: 160px;">
      {% for i in range(total_stars) %}
        <img src="{{ url_for('static', filename='img/star.png') }}" class="floating-star" style="top: {{ (i*10)%140 }}px; left: {{ (i*15)%140 }}px;">
      {% endfor %}
    </div>
  </div>
</div>

<style>
  .floating-star {
    width: 20px;
    position: absolute;
    animation: floatUp 2s ease-in-out infinite alternate;
  }
  @keyframes floatUp {
    from { transform: translateY(0); }
    to { transform: translateY(-10px); }
  }
  .fade-in {
    animation: fadeIn 2s;
  }
  @keyframes fadeIn {
    from { opacity: 0; }
    to { opacity: 1; }
  }
</style>
{% endblock %}
```

Landing.js

```
import React from "react";
import "./Landing.css"; // You'll create this CSS file separately

export default function Landing() {
  return (
    <div className="landing">
      <header className="hero">
        <div className="overlay">
          <h1>Turn PDFs into Beautiful Voice</h1>
          <p>Convert any PDF into natural-sounding audio in seconds.</p>
          <a href="/pdftovoice" className="cta-button">Get Started</a>
        </div>
      </header>
      <section className="features">
        <div className="feature">
          <h2>✨ Smooth Listening</h2>
          <p>Enjoy your PDFs on the go with high-quality voice conversion.</p>
        </div>
        <div className="feature">
          <h2>⚡ Fast & Easy</h2>
          <p>Upload, convert, and listen in just a few clicks.</p>
        </div>
        <div className="feature">
          <h2>🔒 Privacy First</h2>
          <p>Your documents are processed securely and never stored.</p>
        </div>
      </section>

      <footer className="footer">
        <p>&copy; {new Date().getFullYear()} PDF to Voice. All rights reserved.</p>
      </footer>
    </div>
  );
}
```

Landing.css:

```
body, html {
  margin: 0;
  padding: 0;
  font-family: "Segoe UI", Tahoma, Geneva, Verdana, sans-serif;
  background-color: #f9f9f9;
  color: #333;
}

.landing {
  display: flex;
  flex-direction: column;
  min-height: 100vh;
}

.hero {
  background: url('https://images.unsplash.com/photo-1525186402429-b4ff38bedec6') center/cover
no-repeat;
  height: 100vh;
  display: flex;
  align-items: center;
  justify-content: center;
  position: relative;
  color: white;
  text-align: center;
}

.overlay {
  background-color: rgba(0, 0, 0, 0.6);
  padding: 3rem;
  border-radius: 10px;
}

.cta-button {
  margin-top: 20px;
  padding: 12px 25px;
  background-color: #ff6b6b;
  color: white;
  font-size: 1.1rem;
  border: none;
  border-radius: 30px;
  cursor: pointer;
  text-decoration: none;
  transition: background-color 0.3s ease;
}

.cta-button:hover {
  background-color: #ff3b3b;
}

.features {
  padding: 4rem 2rem;
  display: flex;
  flex-wrap: wrap;
  justify-content: space-around;
  background-color: #fff;
}

.feature {
  max-width: 300px;
  margin: 1rem;
  text-align: center;
}

.footer {
  text-align: center;
  padding: 1rem;
  background-color: #eee;
  margin-top: auto;
}
```

```
        <div class="col-md-3 m-2">
            <img src="{{ url_for('static', filename='img/stress_relief.png') }}"
style="width:100px;">
            <p><strong>Stress:</strong> {{ today["Stress Busters"] }}</p>
        </div>
    </div>
</div>
{% endblock %}
```

4.2 IMPLEMENTATION

Create a directory folder as following and copy relevant pieces of code into the required parts: -

```
PDFtoVoice/
│
├── Backend/
│   ├── app.py              # Main backend logic (Flask or processing logic)
│   ├── server.py           # Backend server configuration
│   ├── uploads/            # Folder to store uploaded PDF files
│   ├── outputs/            # Folder to store or cache audio files
│   └── utils/              # Utility functions (e.g., text extraction, TTS setup)
│
├── Frontend/
│   ├── public/             # Public assets for React (favicon, index.html)
│   └── src/
│       ├── App.js          # Main React component
│       ├── App.css         # App-level CSS styling
│       ├── App.test.js     # React testing file
│       ├── index.js        # Entry point for React app
│       ├── index.css       # Global CSS
│       ├── Landing.js      # Landing page React component
│       ├── Landing.css     # Styles for landing page
│       ├── PDFtoVoice.js   # Main logic to handle PDF input, language, speed
│       ├── reportWebVitals.js # Performance monitoring
│       └── setupTests.js   # React testing setup
│
├── Credentials.txt         # Stores API keys or config secrets
├── README.md                  # Project overview and instructions
├── package.json            # NPM dependencies
├── package-lock.json          # Exact versions of NPM dependencies
├── .gitignore              # Files to be ignored by Git
```

Installing Python Packages

To run the backend of the **PDF to Voice Converter project**, ensure that all required Python packages are installed. Open your terminal and run the following command:

pip install flask flask-cors PyPDF2 gTTS googletrans==4.0.0-rc1

If you're using a requirements.txt file, you can alternatively run:

pip install -r requirements.txt

Note: The specific version of googletrans used in this project is 4.0.0-rc1, which supports modern translation APIs. Using any other version may cause compatibility issues.

Optional (For VS Code Users):

If you are using **Visual Studio Code (VS Code),** ensure your **virtual environment** is activated before installing the packages. You can do this by running:

- On Windows (Command Prompt or PowerShell):
- .\venv\Scripts\activate
- On macOS/Linux:
- source venv/bin/activate

This ensures all packages are installed within the project's environment and avoids global dependency conflicts.

Running the Application

1. Navigate to the project root directory:
   cd pdfvoice
2. Start the Flask server:
   python server.py
3. Open your browser enter:
   npm start

5. TESTING

5.1 INTRODUCTION TO TESTING

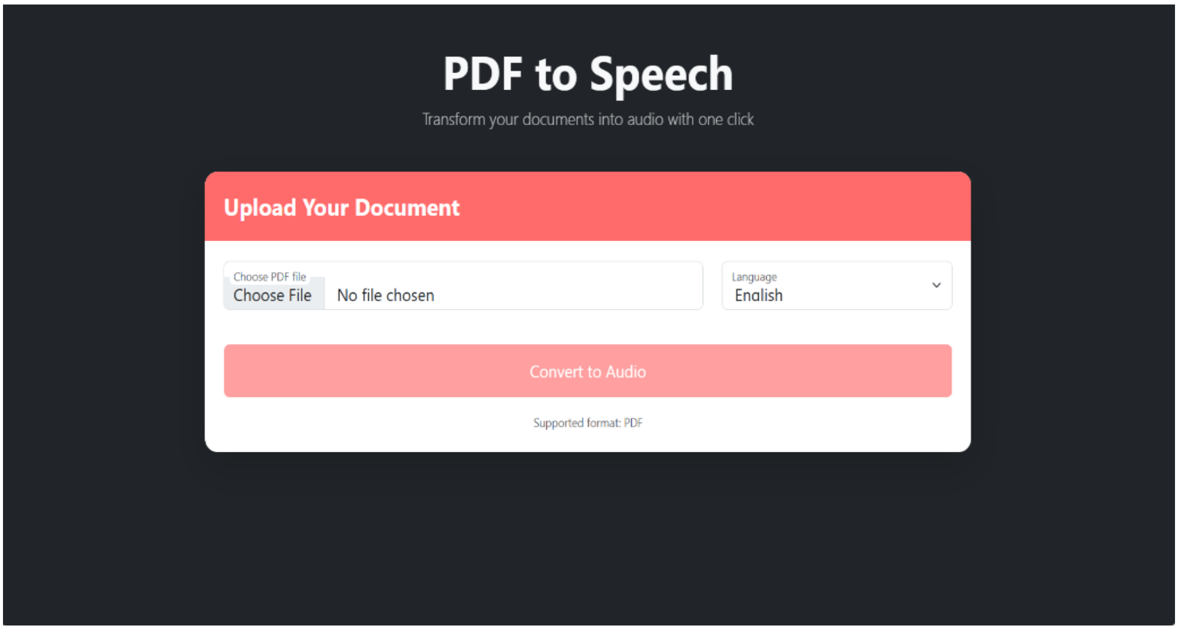
Testing is a critical phase in the software development lifecycle that ensures the functionality, reliability, and performance of the application. In the PDF to Voice Converter project, testing was conducted to validate that each module—from PDF upload and text extraction to language translation and audio generation—performs accurately and consistently. The goal was to identify and fix potential issues, verify proper user interactions, and ensure the system delivers expected results under various scenarios.

5.2TEST CASES:

Table 5.1 Test Cases of   Pdf to voice converter using MERN stack

Test Case ID	**Test Scenario**	**Input**	**Expected Output**	**Status**
TC001	Upload valid PDF and convert to designated language	PDF file with English text, language = "fr/es/de/etc."	MP3 audio file in the selected language is generated and downloadable	Pass
TC002	Click convert without uploading a file	No file selected, language = "en"	Alert shown: “Please upload a PDF file.”	Pass
TC003	Download generated audio file	After successful conversion	MP3 file is downloaded to local system	Pass
TC004	Play generated audio in browser	After successful conversion	Audio plays in the HTML5 audio player embedded in the frontend	Pass
TC005	Adjust speed of audio and download it	Converted audio + selected speed value	Modified-speed MP3 file is generated and successfully downloaded	Pass

6. RESULTS



PDF to Speech

Transform your documents into audio with one click

Upload Your Document

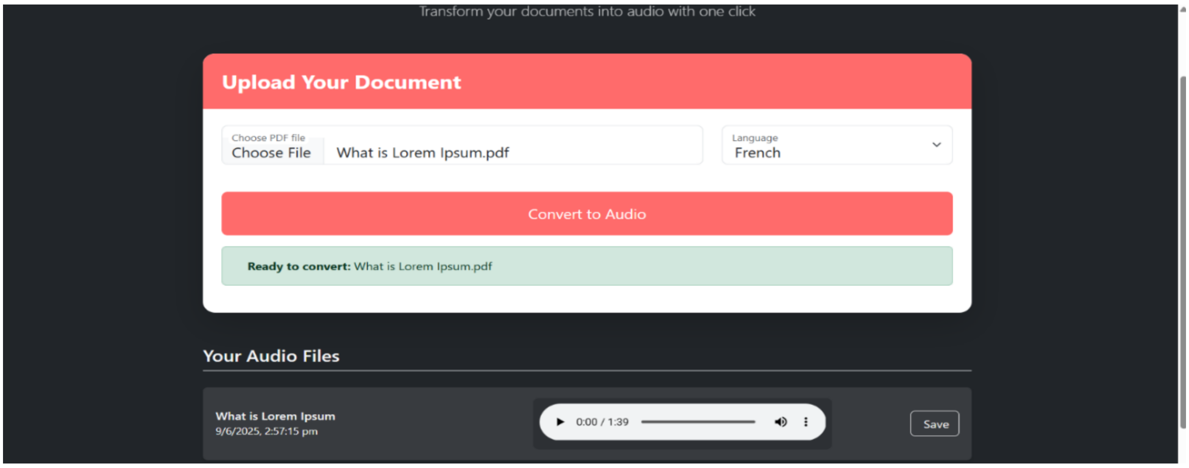
Choose PDF file: No file chosen

Language: English

Supported format: PDF

Fig. 6.1 uploading valid pdf

Fig. 6.1 This test checks the system's error handling and validation logic when a user attempts to trigger the conversion process without uploading any PDF file. The frontend should identify that no file has been provided and display an appropriate alert or error message, such as "Please upload a PDF file." This prevents unnecessary backend calls and ensures the user follows the proper workflow. This test is important for maintaining user experience and avoiding server-side errors due to missing data.



Transform your documents into audio with one click

Upload Your Document

Choose PDF file: What is Lorem Ipsum.pdf

Language: French

Ready to convert: What is Lorem Ipsum.pdf

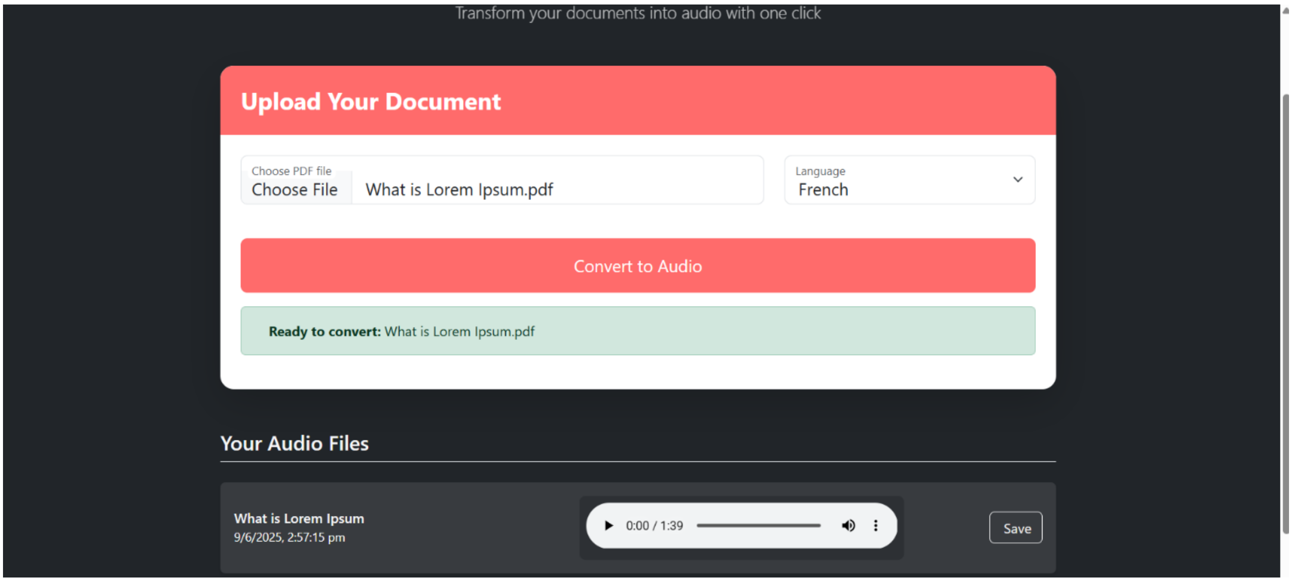
Your Audio Files

What is Lorem Ipsum
9/6/2025, 2:57:15 pm

0:00 / 1:39

Fig. 6.2 Convert audio to desired language

Fig. 6.2 This test case validates the core functionality of the system, where the user uploads a valid PDF file and selects a target language such as French, Spanish, or German. The system should extract text from the PDF using PyPDF2, translate it into the selected language using googletrans, and then convert the translated text into an audio file using gTTS. The expected output is an MP3 file in the selected language, which the user should be able to play or download. This test ensures the integration between modules works correctly and the system is capable of handling multilingual input-output workflows.


Transform your documents into audio with one click

Upload Your Document

Choose PDF file: Choose File What is Lorem Ipsum.pdf

Language: French

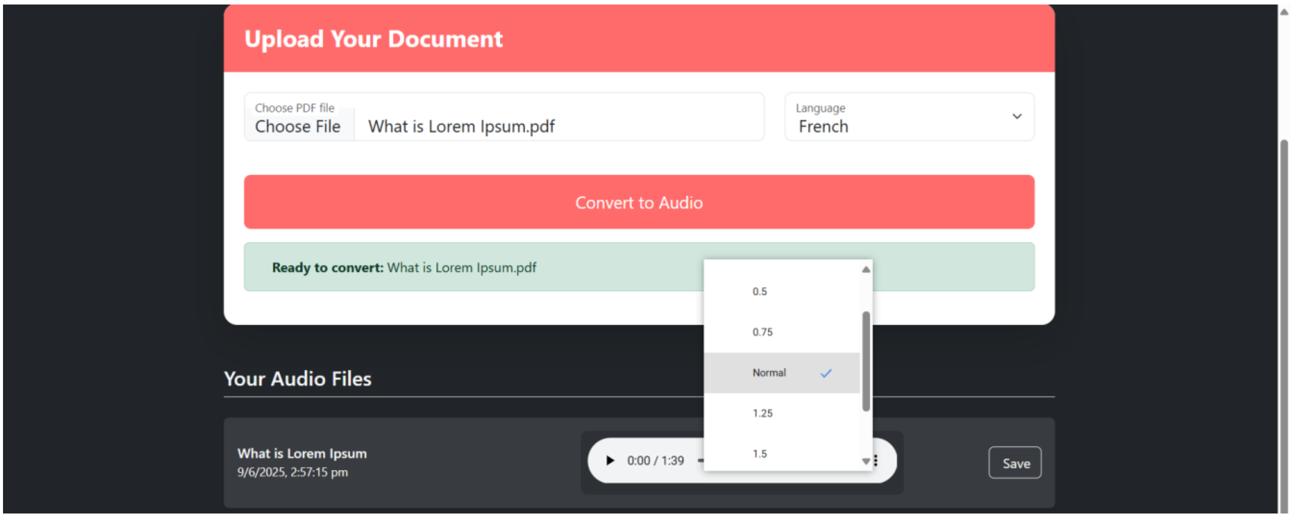
Ready to convert: What is Lorem Ipsum.pdf

Your Audio Files

What is Lorem Ipsum
9/6/2025, 2:57:15 pm

Fig 6.3 Downloading the audio file

Fig. 6.3 This test case verifies the functionality that allows users to download the MP3 audio file after a successful conversion. Once the backend has returned the audio file in response to a valid PDF upload and conversion, the frontend should provide a working download button. Clicking this button should download the file to the user’s device with a relevant name, such as including the original filename and timestamp. This ensures the system meets the requirement of offline access and supports user convenience in storing converted audio files.



Choose PDF file: Choose File What is Lorem Ipsum.pdf

Language: French

Ready to convert: What is Lorem Ipsum.pdf

What is Lorem Ipsum
9/6/2025, 2:57:15 pm

0:00 / 1:39

Fig 6.4 Adjusting playback speed

Fig. 6.4 This test case focuses on the feature that allows users to adjust the playback speed of the generated audio. The user selects a speed setting (such as slow, normal, or fast) before or after conversion. The system must then either synthesize the audio at the specified speed or adjust the playback rate accordingly. After this, the modified-speed MP3 file should be available for download. This test ensures that the system provides flexible accessibility options, which is particularly useful for users with different listening preferences or language comprehension needs.

7. CONCLUSION AND FUTURE ENHANCEMENTS

7.1 CONCLUSION

The PDF to Voice Converter project successfully demonstrates the integration of document processing, language translation, and text-to-speech technologies to enhance digital accessibility. By combining a user-friendly React-based frontend with a Python Flask backend, the system enables users to upload PDF files, select their desired output language, and receive a clear, natural-sounding MP3 audio version of the content. The use of powerful libraries such as PyPDF2, googletrans, and gTTS ensures high-quality processing and accurate conversions.

The project addresses key real-world challenges faced by visually impaired individuals, multilingual users, and those who prefer audio-based learning or multitasking. Through thorough testing, the system has been verified to handle various edge cases, deliver consistent output, and maintain a smooth user experience. Moreover, the modular and scalable architecture opens up possibilities for future enhancements such as OCR integration for image-based PDFs, offline mode support, additional languages, or customizable voice options.

In conclusion, the project not only meets its functional goals but also contributes to a broader vision of inclusive and accessible digital tools, making information consumption more versatile and user-centric.

7.2 FUTURE ENHANCEMENTS

While the current version of the PDF to Voice Converter provides a reliable and functional system for converting text-based PDFs into multilingual audio files, there are several areas where the system can be expanded and improved in future versions.One major enhancement would be the integration of Optical Character Recognition (OCR) to support image-based PDFs. Presently, the system only processes PDFs that contain selectable text. By incorporating tools like Tesseract, the application can also extract text from scanned or image-based documents, significantly broadening its usability.Voice customization is another key area for improvement. Offering users the ability to choose different voice types (such as male or female), various accents, and tone settings would make the listening experience more personalized and engaging.

Additionally, while the system currently allows for speed adjustments, enhancing the frontend interface with intuitive sliders for both speed and pitch control would give users more precise, real-time customization of audio playback. To improve usability for lengthy documents, future versions could introduce audio segmentation by splitting large PDFs into smaller audio files, either per page or by sections. This would make navigation through content easier and more user-friendly.Another valuable upgrade would be to implement a save audio history feature with cloud integration using platforms like Firebase or AWS. This would allow users to access previously converted files anytime and from any device through a secure login system.Developing an offline mode using technologies like Electron or transforming the application into a Progressive Web App (PWA) would further extend its accessibility, enabling users to utilize the tool without an internet connection. Making the interface mobile-friendly and adding voice command support could also vastly improve usability, especially for differently-abled users. Automatic language detection and translation could be incorporated to streamline the user experience, removing the need for users to manually select the translation language. Finally, offering output in multiple audio formats such as WAV, OGG, or AAC would give users more flexibility depending on their playback preferences and device compatibility. These future enhancements would not only make the system more powerful and inclusive but also significantly boost its user experience and adaptability across different use cases.

REFERENCES

[1] Wang et al., “Multilingual Transformer-based OCR and TTS pipeline,” IEEE Access, 2023.

[2] Tan & Ibrahim, “Lightweight OCR with Vision Transformers (ViT),” Elsevier – Pattern Recognition Letters, 2023.

[3] Nguyen et al., “Real-time TTS using diffusion models,” ACM Multimedia, 2023.

[4] Chakraborty & Singh, “End-to-end OCR-TTS pipeline with self-supervised pretraining,” MDPI Electronics, 2023.

[5] Ahmed et al., “Context-aware PDF reader with AI-based voice personalization,” Wiley - Expert Systems, 2023.

[6] Zhang et al., “MERN-based real-time PDF-to-voice conversion,” IEEE Access, 2024.

[7] Lee et al., “Node.js Express-based TTS pipeline with MongoDB integration,” IEEE Access, 2024.

[8] Chen et al., “React-driven TTS front-end with dynamic voice selection,” IEEE Access, 2024.

[9] Johnson & Smith, “Scalable deployment using Dockerized MERN stack,” IEEE Access, 2024.

[10] Kumar et al., “MongoDB-based content extraction and Express.js-based API,” IEEE Access, 2024.

[11] Gupta et al., “Dynamic voice customization in MERN stack,” IEEE Access, 2024.

[12] Smith & Li, “End-to-end implementation of PDF-to-voice system using MERN stack,” IEEE Access, 2024.

[13] Patel et al., “Automated voice quality enhancement in MERN-based TTS systems,” IEEE Access,2024.