# Audio-to-Text Transcription App Overview

This web-based application is designed to efficiently convert audio files (MP3 or WAV) into editable text, which users can then download in `.txt` or `.pdf` format. Built using **Streamlit**, a Python framework for building interactive web apps, the tool features a clean, user-friendly interface that supports both file uploads and live audio recordings.

It is ideal for transcribing **meetings**, **lectures**, **podcasts**, or any other spoken content—perfect for students, journalists, researchers, and professionals.

---

## Core Technologies and Libraries

The application leverages several powerful Python libraries:

- **SpeechRecognition**: Converts spoken words into written text using Google's Speech-to-Text API.
- **PyDub**: Processes and converts audio files (e.g., MP3 to WAV).
- **FPDF**: Generates downloadable PDF documents.
- **Streamlit**: Powers the user interface and interactions.
- **audio_recorder_streamlit**: Enables in-browser audio recording.
- **wave**, **tempfile**, **os**, and **urllib**: Handle file operations and temporary storage.

---

## Key Features

1. **Audio Upload & Live Recording**
   Users can upload pre-recorded audio files or record directly within the app.

2. **Multilingual Support**
   Interface and transcription options available in **English** and **Turkish**.

3. **Accurate Transcription**
   Speech is converted into text using a reliable cloud-based API with language-specific models.

4. **Editable Transcript**
   After transcription, users can review and edit the text in-app.

5. **Downloadable Output**
   Final transcripts can be downloaded in `.txt` or `.pdf` format.

6. **PDF Font Handling**
   Automatically downloads the **NotoSans-Regular** font if not already present to support multilingual character sets in PDF output.

7. **Session Persistence**
   The app remembers user settings (language, audio source, transcript) using Streamlit's session state.

8. **Robust Error Handling**
   Descriptive error messages are displayed in case of upload failures, audio issues, or API errors.

---

## Workflow Summary

1. **Language Selection**
   Choose interface and transcription language (e.g., English, Turkish).

2. **Audio Input**
   Upload an audio file or use live recording.

3. **Conversion**
   Audio is converted to WAV (if needed), then transcribed using **SpeechRecognition**.

4. **Editing**
   The transcribed text is shown in an editable text area for user review.

5. **Download**
   The edited transcript can be downloaded as a `.txt` or `.pdf` file.

6. **Error Handling**
   Any errors during processing are caught and displayed with helpful messages.

---

## Code Components Explained

## Importing Libraries

```
import streamlit as st
from audio_recorder_streamlit import audio_recorder
import speech_recognition as sr
from pydub import AudioSegment
from fpdf import FPDF
```

```python
import io, wave, os, urllib.request, tempfile
from datetime import datetime
```

**Explanation:**

- **streamlit as st**: Streamlit is a Python library used to create interactive web applications. The alias `st` is used to call Streamlit functions more concisely (e.g., `st.write()`, `st.button()`).
- **audio_recorder_streamlit**: This library provides a tool for users to record audio directly through their browser. The `audio_recorder` function is used to record audio.
- **speech_recognition as sr**: This library is used for converting audio into text. It leverages Google's Speech Recognition API for transcription.
- **pydub.AudioSegment**: PyDub is used for processing and converting audio files. The `AudioSegment` class can convert audio from one format to another (e.g., from MP3 to WAV).
- **fpdf.FPDF**: This library is used for generating PDF documents. The `FPDF` class enables the creation of a PDF file and allows adding text to it.
- **io, wave, os, urllib.request, tempfile**:
  - **io**: Used for handling file operations and working with byte data.
  - **wave**: Used for working with WAV audio files.
  - **os**: Allows interacting with the operating system for file management tasks like checking file paths or deleting files.
  - **urllib.request**: Used for downloading files from the internet (e.g., font files).
  - **tempfile**: Used for creating temporary files, which is useful for managing files during audio processing or PDF creation.
- **datetime**: Provides functionalities to work with dates and times. It's helpful for timestamping files or saving them with the current date.

---

## Font Setup for PDF

```python
FONT_NAME = "NotoSans-Regular"
FONT_FILE = f"{FONT_NAME}.ttf"
if not os.path.exists(FONT_FILE):
    urllib.request.urlretrieve(

"https://github.com/googlefonts/noto-fonts/raw/main/hinted/ttf/NotoSan
s/NotoSans-Regular.ttf",
        FONT_FILE
    )
```

**Explanation:**

- **FONT_NAME and FONT_FILE**: The application uses the **NotoSans-Regular** font for creating PDF files. The font file is defined as `FONT_NAME` and stored with the `.ttf` extension in `FONT_FILE`.
- **os.path.exists()**: This function checks whether the font file already exists locally. If it does, nothing further is done.
- **urllib.request.urlretrieve()**: If the font file is not found, the script downloads it from the given URL (from Google Fonts) to ensure it is available for PDF creation.

---

## Language Support Structure

```
LANGUAGES = {
    "▯▯ Turkish": { "title": "Ses-Metin Dönüştürücü", ... },
    "▯▯ English": { "title": "Audio-Text Converter", ... }
}
```

**Explanation:**

- **LANGUAGES**: This dictionary handles the multilingual support for the application. Users can choose the language for the interface (e.g., Turkish or English).
  - Each language has a corresponding title and labels for various UI elements (e.g., "Input Type", "Options").
  - For example, **"▯▯ Turkish"** will display the title **"Ses-Metin Dönüştürücü"** and **"▯▯ English"** will display **"Audio-Text Converter"**.
- This structure allows the app to dynamically change the language of the UI based on the user's selection.

---

## PDF Generation

```
def create_pdf(text):
    pdf = FPDF()
    pdf.add_page()
    pdf.add_font(FONT_NAME, '', FONT_FILE, uni=True)
    pdf.set_font(FONT_NAME, size=12)
    pdf.multi_cell(0, 10, text)
    return pdf.output(dest='S')
```

**Explanation:**

- **FPDF()**: Initializes a new PDF document.
- **add_page()**: Adds a new page to the PDF.
- **add_font()**: Adds a custom font to the PDF. In this case, **NotoSans-Regular** font is used. The font is added using the previously downloaded font file.
- **set_font()**: Sets the font and size for the PDF content. The font is set to **NotoSans-Regular** at 12-point size.
- **multi_cell()**: Adds multi-line text to the PDF. If the text exceeds one line, it wraps to the next line.
- **pdf.output(dest='S')**: Generates the PDF and returns it as a byte object, which can then be sent to the user.

---

## Session Management

```
if 'recording' not in st.session_state:
    st.session_state.recording = False
...
```

**Explanation:**

- **st.session_state**: This is Streamlit's session management system. It allows data to persist across user interactions and prevents data from being lost when the user refreshes the page.
- This block of code checks whether certain variables (e.g., whether a recording is in progress, the selected language, and the transcript) exist in the session state. If they don't, it initializes them with default values (e.g., `False` for `recording`).

---

## Audio Conversion & Transcription

### Convert Audio to WAV

```
def convert_to_wav(uploaded_file):
    ...
```

**Explanation:**

- This function takes an uploaded audio file and converts it into a **WAV** format, which is a standard format for audio processing. The PyDub library is used to perform the conversion, ensuring that the audio is in the correct format for transcription.

---

## Transcribe the Audio

```
def process_audio(file_path, language):
    ...
```

**Explanation:**

- This function processes the audio file and converts it into text using the **SpeechRecognition** library.
- It uses the **recognize_google()** function to transcribe the audio to text, specifying the language (from the **LANGUAGES** dictionary) to ensure the transcription is accurate.

---

## Main Functionality Flow

```
def main():
    lang = st.sidebar.selectbox("□ Language", list(LANGUAGES.keys()))
    ...
    input_type = st.radio(current_lang["input_type"],
current_lang["options"], horizontal=True)
    ...
    if audio_path or st.session_state.file_uploaded:
        text = process_audio(audio_path, lang)
    ...
    edited_text = st.text_area(current_lang["edit_text"],
st.session_state.transcript, height=300)
```

**Explanation:**

- **lang = st.sidebar.selectbox()**: A sidebar dropdown allows users to select their preferred language. The language choice impacts the entire UI, showing labels and buttons in the selected language.

- **input_type = st.radio()**: A radio button is provided for users to choose between uploading an audio file or recording audio directly. The available options are dynamically generated based on the selected language.

- **process_audio()**: If the user uploads a file or records audio, this function is used to process the audio file and convert it to text.

- **st.text_area()**: After the transcription, a text box is provided for the user to review and edit the transcribed text. The user can modify any errors before saving or downloading the transcript.

---

## Conclusion

This app streamlines the process of converting spoken content into accurate, editable, and shareable text. With built-in recording, multilingual support, and multiple download formats, it's a practical tool for anyone needing high-quality transcriptions—whether you're a journalist capturing interviews, a student recording lectures, or a professional documenting meetings.