Programming for Artificial Intelligence

**Name: Sara hanif**

**Roll number:082**

**PAI Project: Music Lyrics Generator Web App**

# 1. Introduction

The Music Lyrics Generator is a deep learning-based web application that allows users to generate thematic song lyrics and corresponding music tracks. Built using Python, Flask, and NLP/audio generation models, this app offers an intuitive platform for lyric creation, audio synthesis, and speech conversion. It combines AI text generation and music production to assist artists, learners, and enthusiasts in creative songwriting.

# 2. Tools & Technologies

- Language: Python 3.x  
- Web Framework: Flask  
- AI Models: HuggingFace Transformers, Audiocraft  
- Audio Synthesis: pyttsx3 (Text-to-Speech)  
- Front-End: HTML, CSS  
- Environment Management: Virtual Environment (myenv)

# 3. Project Structure & File Overview

## 3.1 File Descriptions

- app.py:  
 Main Flask backend that handles routing, form input, lyrics generation, music generation, text-to-speech, and rendering HTML.

from flask import Flask, render\_template, request, send\_from\_directory

from generate\_lyrics import generate\_lyrics

from generate\_music import generate\_music

import os

import pyttsx3

app = Flask(\_\_name\_\_)

OUTPUT\_DIR = "static/generated"

os.makedirs(OUTPUT\_DIR, exist\_ok=True)

@app.route("/", methods=["GET", "POST"])

def index():

    lyrics = ""

    if request.method == "POST":

        custom\_lyrics = request.form.get("custom\_lyrics", "").strip()

        theme = request.form.get("theme", "").strip()

        if custom\_lyrics:

            lyrics = custom\_lyrics

        elif theme:

            lyrics = generate\_lyrics(theme)

        else:

            lyrics = "No lyrics provided."

        # Save lyrics to text file

        lyrics\_path = os.path.join(OUTPUT\_DIR, "lyrics.txt")

        with open(lyrics\_path, "w") as f:

            f.write(lyrics)

        # Generate music

        music\_path = os.path.join(OUTPUT\_DIR, "music.wav")

        generate\_music(lyrics, music\_path)

        # Text-to-speech

        engine = pyttsx3.init()

        tts\_path = os.path.join(OUTPUT\_DIR, "tts.wav")

        engine.save\_to\_file(lyrics, tts\_path)

        engine.runAndWait()

    return render\_template("index.html", lyrics=lyrics)

@app.route("/download/<filename>")

def download\_file(filename):

    return send\_from\_directory(OUTPUT\_DIR, filename, as\_attachment=True)

if \_\_name\_\_ == "\_\_main\_\_":

    app.run(debug=True)

- generate\_lyrics.py:  
 Contains functions to generate lyrics based on a theme using NLP models.

from transformers import pipeline

def generate\_lyrics(topic, theme):

    generator = pipeline("text-generation", model="gpt2")

    prompt = f"Write a {theme} song about {topic}:\n"

    result = generator(prompt, max\_length=100, do\_sample=True, temperature=0.9)

    lyrics = result[0]['generated\_text']

    return lyrics.strip()

- generate\_music.py:  
 Produces a music file based on the generated lyrics using the Audiocraft model.

import torchaudio

from audiocraft.models import MusicGen

from audiocraft.data.audio import audio\_write

def generate\_music(lyrics, output\_path):

    model = MusicGen.get\_pretrained('facebook/musicgen-small')

    model.set\_generation\_params(duration=10)

    wav = model.generate([lyrics])

    audio\_write(output\_path.replace(".wav", ""), wav[0].cpu(), model.sample\_rate, strategy="loudness")

- index.html:  
 Frontend interface for users to input custom lyrics or a theme and view results.

<html>

<head>

    <title>Music Lyrics Generator</title>

    <link rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}">

    <link href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;500;700&display=swap" rel="stylesheet">

</head>

<body>

    <div class="container">

        <h1>🎵 Music Lyrics Generator</h1>

        <form method="post">

            <label>Choose a theme:</label>

            <select name="theme">

                <option value="">-- Select Theme --</option>

                <option value="love">Love</option>

                <option value="sad">Sad</option>

                <option value="happy">Happy</option>

                <option value="inspirational">Inspirational</option>

            </select>

            <label>Or write your own lyrics:</label>

            <textarea name="custom\_lyrics" rows="8" placeholder="Write your lyrics here..."></textarea>

            <input type="submit" value="🎶 Generate Music">

        </form>

        {% if lyrics %}

            <div class="result">

                <h2>✨ Generated Lyrics</h2>

                <pre>{{ lyrics }}</pre>

                <div class="downloads">

                    <a href="{{ url\_for('download\_file', filename='lyrics.txt') }}">📄 Download Lyrics (.txt)</a>

                    <a href="{{ url\_for('download\_file', filename='music.wav') }}">🎧 Download Music (.wav)</a>

                    <a href="{{ url\_for('download\_file', filename='tts.wav') }}">📢 Download Voice Note (.wav)</a>

                </div>

            </div>

        {% endif %}

    </div>

</body>

</html>

- style.css:  
 CSS file providing styling for the web interface.

body {

    font-family: 'Poppins', sans-serif;

    background: linear-gradient(to right, #f8f9fa, #e0eafc);

    color: #333;

    margin: 0;

    padding: 0;

}

.container {

    max-width: 800px;

    margin: 40px auto;

    background: #fff;

    padding: 30px;

    border-radius: 15px;

    box-shadow: 0 8px 20px rgba(0, 0, 0, 0.1);

}

h1 {

    color: #2c3e50;

    text-align: center;

    margin-bottom: 20px;

    font-size: 2.5rem;

}

form {

    display: flex;

    flex-direction: column;

    gap: 15px;

}

label {

    font-weight: 500;

    margin-bottom: 5px;

}

select, textarea {

    padding: 12px;

    font-size: 1rem;

    border-radius: 8px;

    border: 1px solid #ccc;

    font-family: 'Poppins', sans-serif;

    resize: vertical;

}

input[type="submit"] {

    background-color: #5b73e8;

    color: white;

    border: none;

    padding: 14px;

    font-size: 1rem;

    border-radius: 8px;

    cursor: pointer;

    transition: background-color 0.3s;

}

input[type="submit"]:hover {

    background-color: #4051c7;

}

.result {

    margin-top: 30px;

}

.result h2 {

    color: #34495e;

    font-size: 1.8rem;

    margin-bottom: 10px;

}

pre {

    background: #f4f4f4;

    padding: 20px;

    border-left: 6px solid #5b73e8;

    border-radius: 10px;

    white-space: pre-wrap;

    word-wrap: break-word;

    font-size: 1rem;

    font-family: 'Courier New', monospace;

}

.downloads a {

    display: inline-block;

    margin-top: 12px;

    margin-right: 15px;

    color: #5b73e8;

    text-decoration: none;

    font-weight: 600;

    transition: color 0.2s;

}

.downloads a:hover {

    text-decoration: underline;

    color: #34495e;

}

- requirements.txt:  
 Lists all required Python libraries including Flask, torch, transformers, pyttsx3, and Audiocraft.  
Flask

torch

transformers

pyttsx3

audiocraft

- static/:  
 Stores generated output files such as lyrics (lyrics.txt), music (music.wav), and speech (tts.wav).  
  
- templates/:  
 Contains the index.html file for frontend rendering.

## 3.2 File Connection & Execution Order

Startup:

python app.py

Process Flow:

1. Flask app loads.  
2. User accesses the homepage with an input form.  
3. User submits:  
 - A theme → lyrics are generated.  
 - Custom lyrics → used as input.  
4. Lyrics are saved to a file.  
5. Music is generated using generate\_music.  
6. Lyrics are converted to speech using pyttsx3.  
7. Output is displayed and available for download.

# 4. How It Works

## Homepage Load:

Shows an input form for entering lyrics or a theme.  
Displays the output area after submission.

## User Interaction:

User submits either:  
- A theme → NLP model generates lyrics.  
- Custom lyrics → directly used.

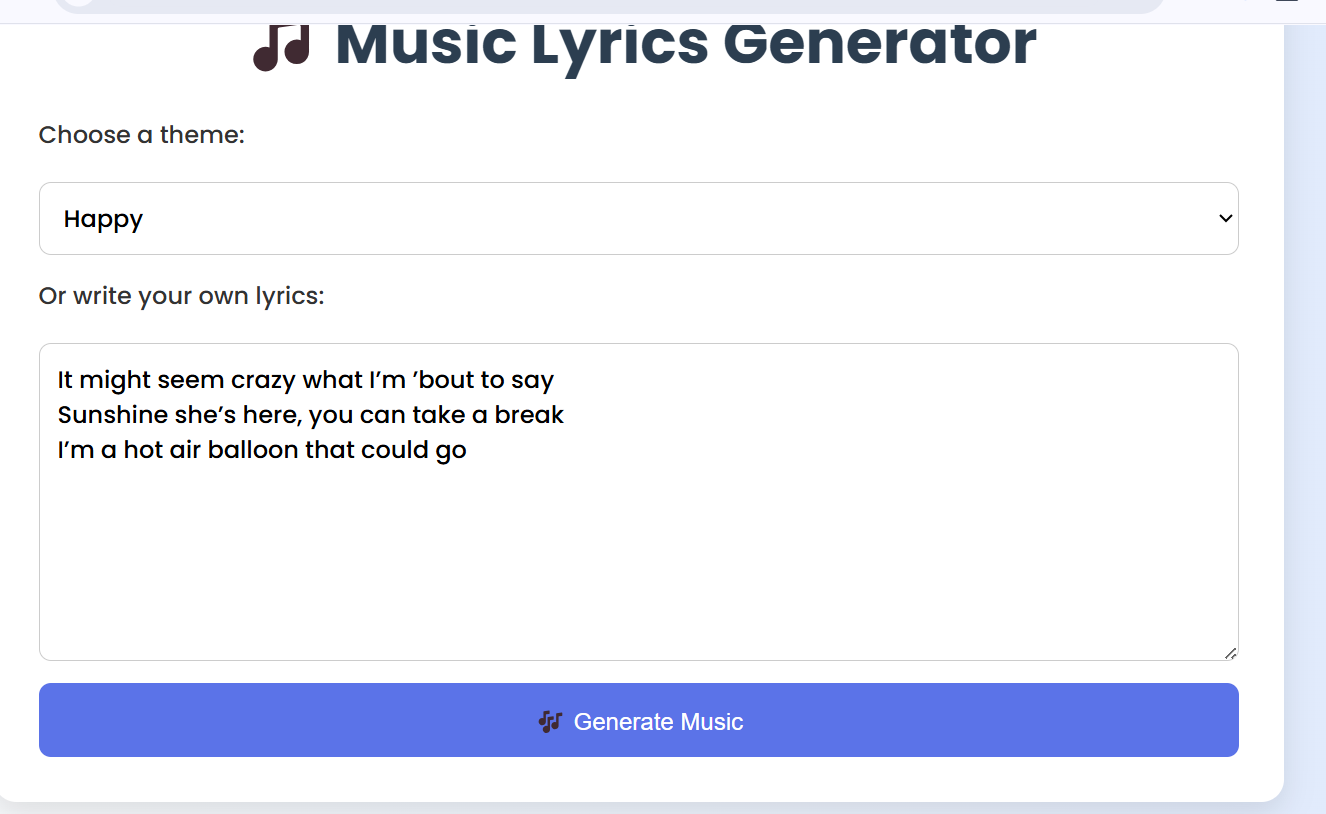
## Backend Logic:

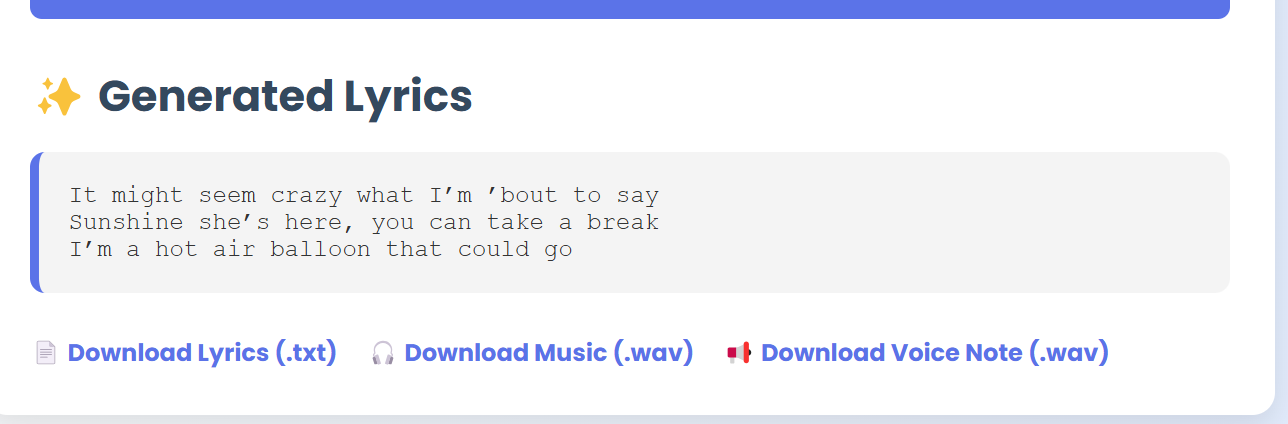
- Lyrics saved to static/generated/lyrics.txt.  
- generate\_music.py uses lyrics to create a .wav file.  
- pyttsx3 converts lyrics into a .wav speech file.

# 5. Results

## Example Input:

Theme: happy





## Output Files:

- lyrics.txt: Generated lyrics text.  
- music.wav: AI-generated background music.  
- tts.wav: Spoken version of the lyrics.

## UI Features:

- Simple, responsive design.  
- Form submission with live output generation.  
- File download functionality.

# 6. Conclusion

The Music Lyrics Generator demonstrates the creative potential of AI in the arts. Combining text generation, music synthesis, "  
and speech technologies, it enables dynamic content creation in a user-friendly web interface. Future enhancements may include:  
  
- Fine-tuning models on custom lyrics/music datasets.  
- Allowing genre-specific generation.  
- Integrating more realistic TTS engines or music generators.