CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

NAAC GRADE A+
Accredited University

Python Project On

"Python Music Player Application"

Submitted by: Prabhakar Kumar

UID: 24MCA20071

Section/Group: 2/A

Submitted To: Ms.Pawandeep Sharma

Course: MCA

Date Of Submission: 25/10/2024

July 2024 – November 2024

University Institute Of Computing

Chandigarh University

Mohali, Punjab

Project: Python Music Player Application.

1. Introduction

This project is a simple Music Player application developed using Python. It leverages the Tkinter library to create the Graphical User Interface (GUI) and pygame.mixer to handle audio playback. The player allows users to load a music directory, play, pause, resume, stop, and navigate between songs.

2. Objective

The primary objective of this project is to demonstrate how to build a lightweight music player with essential features using Python's GUI and multimedia libraries. It aims to provide users with an intuitive interface to browse and play audio files conveniently.

3. Modules and Technologies Used

- **Python Libraries:-**
  - Tkinter: Used to create the GUI for the music player.
  - pygame.mixer: Provides audio playback functionalities.
  - os: Handles file operations like changing directories and listing audio files.
  - filedialog and messagebox: Helps in opening directories and displaying error messages.

4. Features of the Application

1. Play a Song: Plays the currently selected song from the playlist.
2. Stop Song: Stops the playback of the current song.
3. Pause & Resume: Pauses and resumes the song.
4. Next Song: Plays the next song in the playlist.
5. Load Directory: Allows users to load a directory containing music files.
6. Playlist Management: Displays the list of available songs for easy navigation.

5. Code Functionality Overview

5.1 Initialization:-

- mixer.init() initializes the audio player from the pygame library.
- The root window and frames are created using Tkinter.

5.2 Song Management Functions:-

- play_song(): Loads and plays the selected song from the playlist.
- stop_song(): Stops the playback.
- pause_song(): Pauses the current song.
- resume_song(): Resumes the paused song.
- next_song(): Automatically selects and plays the next song in the playlist.

5.3 Directory Loading :-

- load(): Opens a dialog box for users to choose a directory and loads all valid audio files (e.g., .mp3, .wav, .ogg) into the playlist.

5.4 GUI Layout:-

- The GUI consists of:
    1. Song Frame: Displays the currently playing song.
    2. Control Buttons Frame: Contains buttons for Play, Pause, Resume, Next, and Load Directory.
    3. Playlist Frame: Displays the list of songs in the chosen directory.
    4. Status Label: Displays the current state of the player (e.g., "Playing", "Paused").

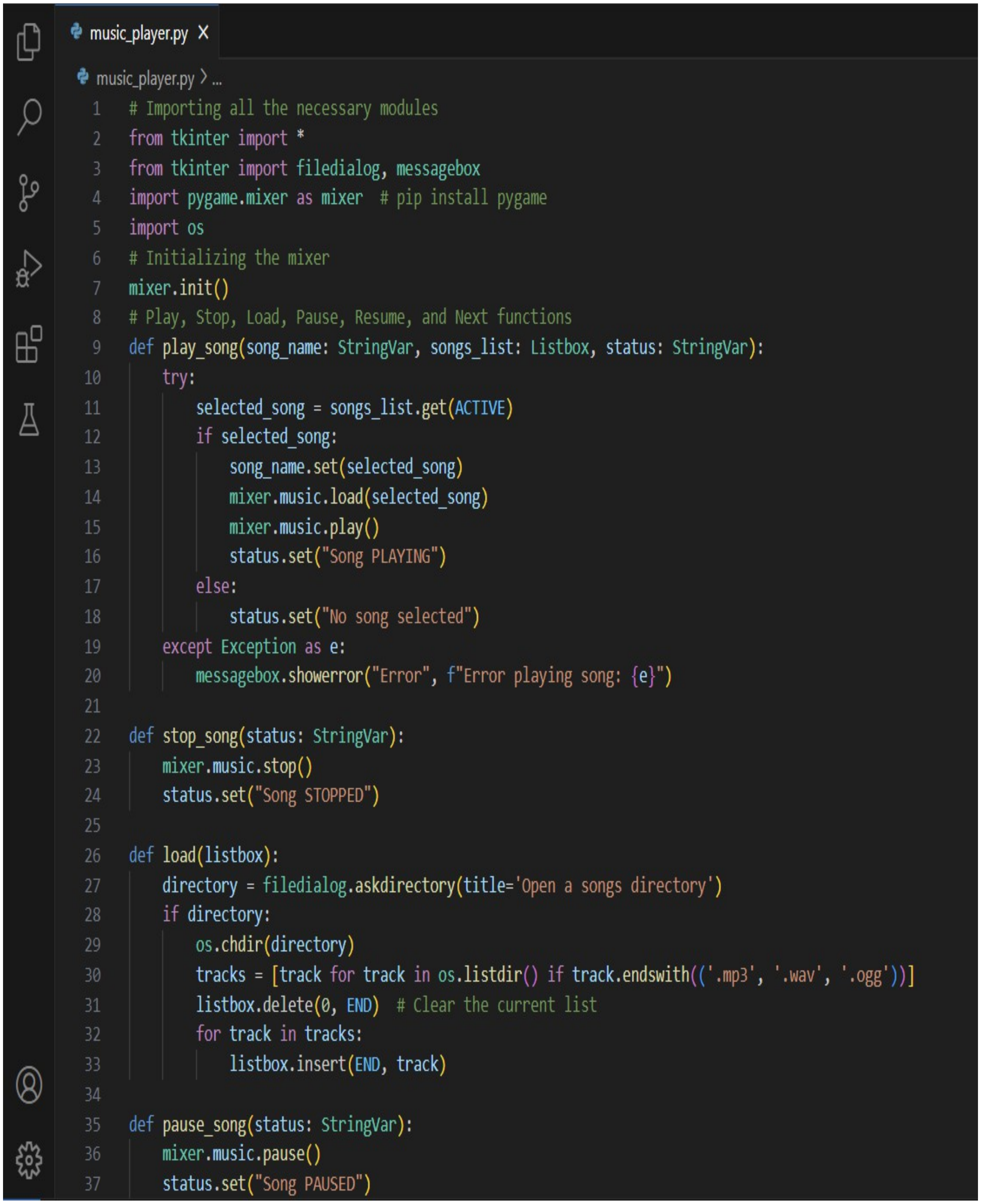
6. User Interface Design:-

- Frames and Buttons: The application uses Tkinter's LabelFrame and Button widgets to organize the controls.
- Playlist: Implemented using the Listbox widget, allowing users to select and manage songs.
- Color Themes: Custom colors are applied to different components for better visual distinction.

7. Challenges Faced:-

- Handling errors when the user selects an invalid file or directory.
- Implementing the "Next Song" feature to loop back to the first song after the last one is played.
- Ensuring smooth pause and resume functionality using pygame.mixer.

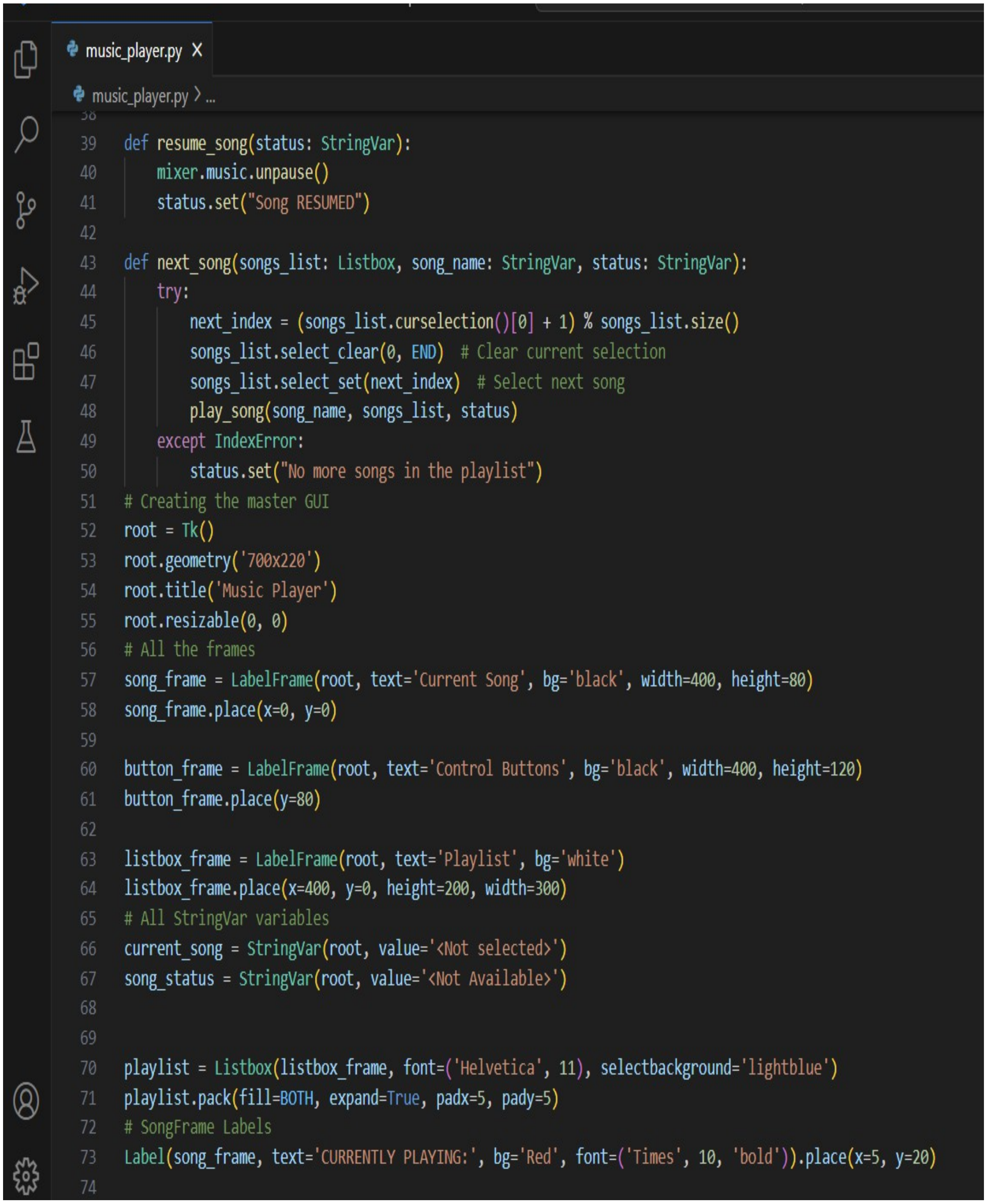
Code:-



music_player.py X

music_player.py > ...

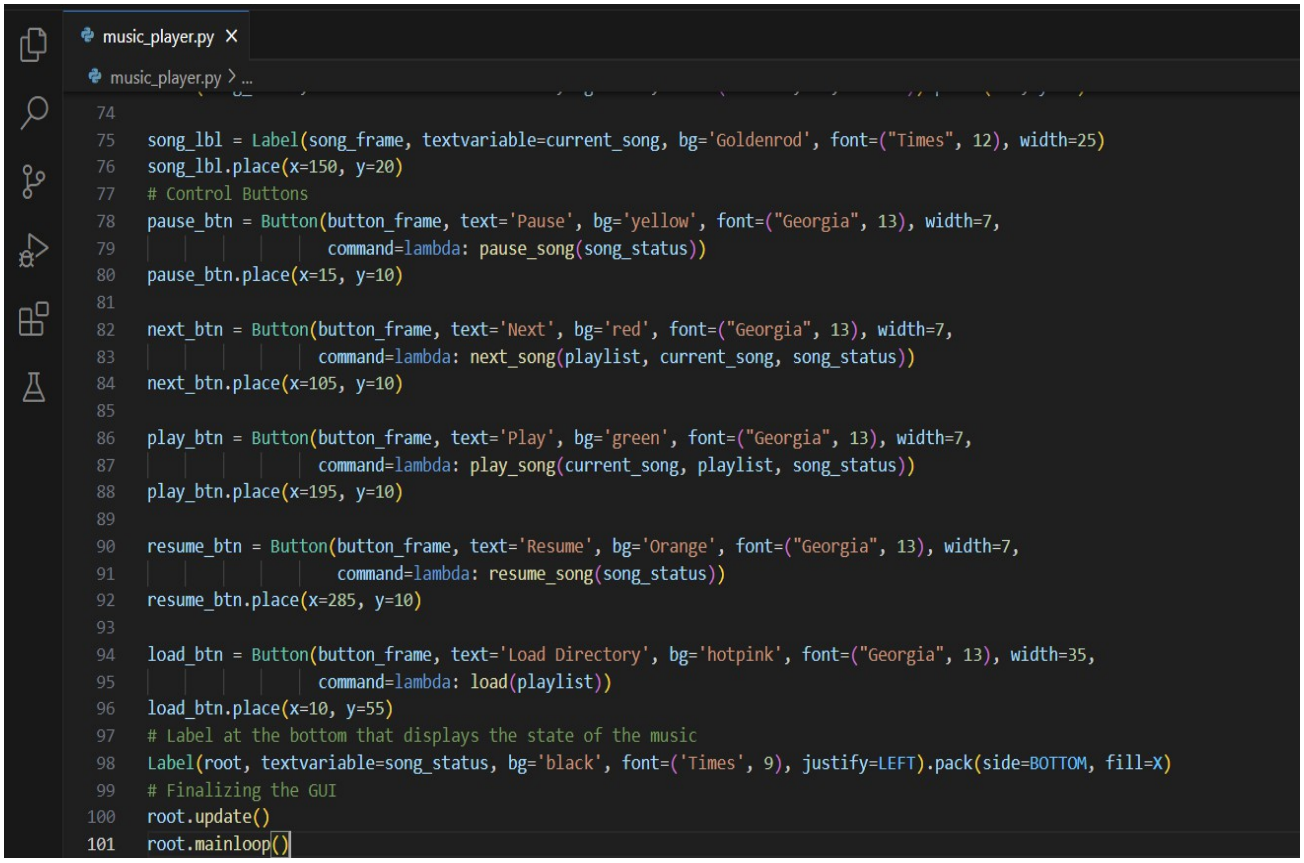
```python
# Importing all the necessary modules
from tkinter import *
from tkinter import filedialog, messagebox
import pygame.mixer as mixer  # pip install pygame
import os
# Initializing the mixer
mixer.init()
# Play, Stop, Load, Pause, Resume, and Next functions
def play_song(song_name: StringVar, songs_list: Listbox, status: StringVar):
    try:
        selected_song = songs_list.get(ACTIVE)
        if selected_song:
            song_name.set(selected_song)
            mixer.music.load(selected_song)
            mixer.music.play()
            status.set("Song PLAYING")
        else:
            status.set("No song selected")
    except Exception as e:
        messagebox.showerror("Error", f"Error playing song: {e}")

def stop_song(status: StringVar):
    mixer.music.stop()
    status.set("Song STOPPED")

def load(listbox):
    directory = filedialog.askdirectory(title='Open a songs directory')
    if directory:
        os.chdir(directory)
        tracks = [track for track in os.listdir() if track.endswith(('.mp3', '.wav', '.ogg'))]
        listbox.delete(0, END)  # Clear the current list
        for track in tracks:
            listbox.insert(END, track)

def pause_song(status: StringVar):
    mixer.music.pause()
    status.set("Song PAUSED")
```



music_player.py

music_player.py > ...

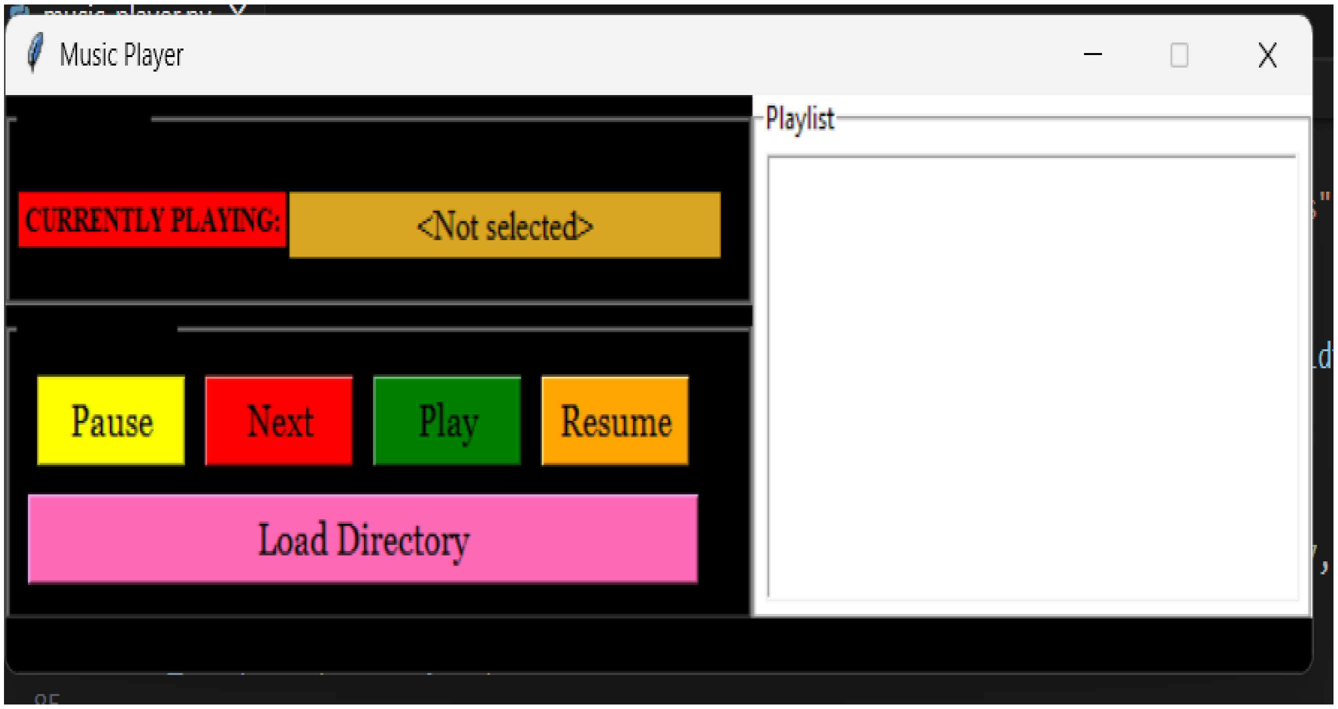
```python
def resume_song(status: StringVar):
    mixer.music.unpause()
    status.set("Song RESUMED")

def next_song(songs_list: Listbox, song_name: StringVar, status: StringVar):
    try:
        next_index = (songs_list.curselection()[0] + 1) % songs_list.size()
        songs_list.select_clear(0, END)  # Clear current selection
        songs_list.select_set(next_index)  # Select next song
        play_song(song_name, songs_list, status)
    except IndexError:
        status.set("No more songs in the playlist")
# Creating the master GUI
root = Tk()
root.geometry('700x220')
root.title('Music Player')
root.resizable(0, 0)
# All the frames
song_frame = LabelFrame(root, text='Current Song', bg='black', width=400, height=80)
song_frame.place(x=0, y=0)

button_frame = LabelFrame(root, text='Control Buttons', bg='black', width=400, height=120)
button_frame.place(y=80)

listbox_frame = LabelFrame(root, text='Playlist', bg='white')
listbox_frame.place(x=400, y=0, height=200, width=300)
# All StringVar variables
current_song = StringVar(root, value='<Not selected>')
song_status = StringVar(root, value='<Not Available>')


playlist = Listbox(listbox_frame, font=('Helvetica', 11), selectbackground='lightblue')
playlist.pack(fill=BOTH, expand=True, padx=5, pady=5)
# SongFrame Labels
Label(song_frame, text='CURRENTLY PLAYING:', bg='Red', font=('Times', 10, 'bold')).place(x=5, y=20)

```



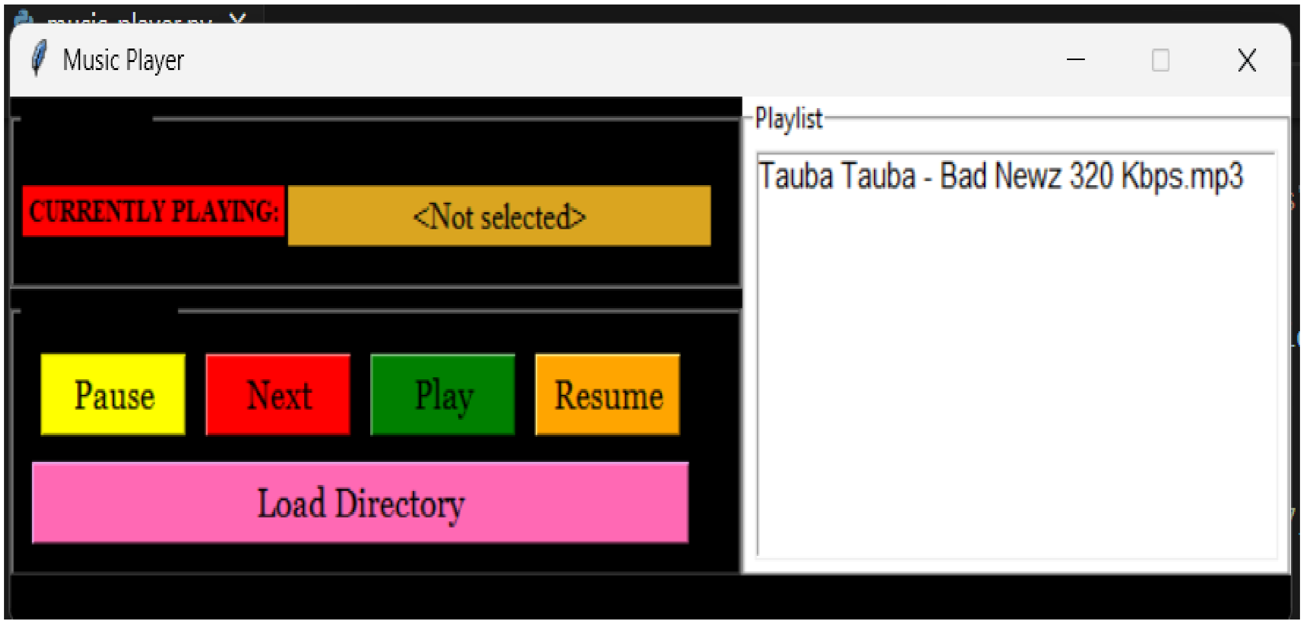
```python

song_lbl = Label(song_frame, textvariable=current_song, bg='Goldenrod', font=("Times", 12), width=25)
song_lbl.place(x=150, y=20)
# Control Buttons
pause_btn = Button(button_frame, text='Pause', bg='yellow', font=("Georgia", 13), width=7,
                    command=lambda: pause_song(song_status))
pause_btn.place(x=15, y=10)

next_btn = Button(button_frame, text='Next', bg='red', font=("Georgia", 13), width=7,
                   command=lambda: next_song(playlist, current_song, song_status))
next_btn.place(x=105, y=10)

play_btn = Button(button_frame, text='Play', bg='green', font=("Georgia", 13), width=7,
                   command=lambda: play_song(current_song, playlist, song_status))
play_btn.place(x=195, y=10)

resume_btn = Button(button_frame, text='Resume', bg='Orange', font=("Georgia", 13), width=7,
                     command=lambda: resume_song(song_status))
resume_btn.place(x=285, y=10)

load_btn = Button(button_frame, text='Load Directory', bg='hotpink', font=("Georgia", 13), width=35,
                   command=lambda: load(playlist))
load_btn.place(x=10, y=55)
# Label at the bottom that displays the state of the music
Label(root, textvariable=song_status, bg='black', font=('Times', 9), justify=LEFT).pack(side=BOTTOM, fill=X)
# Finalizing the GUI
root.update()
root.mainloop()
```

OUTPUT:-

1.Without Song:-



2. With Song:-



Conclusion:-

This project demonstrates a simple yet functional music player in Python. It covers key concepts like **GUI design, event handling, and file management**. The modular approach ensures that users can easily navigate songs and control playback through an intuitive interface.