Name : Cheralathan.P

Course : DADS – RP29 Batch

Milestone-1

Music Player

Project Description:

1. Aim of the Project:

The aim of this project is to develop a simple yet functional music player application using Python and the Pygame library. The primary objective is to enable users to manage and play their music files with basic functionalities such as adding songs, playing, pausing, resuming, restarting, stopping, and adjusting the volume. By implementing this application, we aim to provide a user-friendly tool that integrates music playback controls with a straightforward interface, allowing users to interact with their music collection seamlessly.

2. Problem Statement:

Design and implement a music player application that allows users to manage and play their music files. The application should have the following features:

1. **Add songs to a playlist:** Not all the people listen to online media player application, this enables user to import the songs which are stored in local storage.

2. **Play, pause, resume, restart, and stop songs:** Most of the python libraries like playsound does't have the function to manipulate the audio files, winsound library only lets you manipulate .wav file and not .mp3 files.

3. **Skip to a specific time in the song:** Skipping to certain position of the audio track is achieved by only entering the values in seconds and not in minutes.

4. **Display the current song and artist:** Till now the project displays the title of the current song and Artist who composed the song and need to find a way to implement the playback timer.

3. Project Description:

This project involves the development of a basic music player application utilizing Python programming and the Pygame library. The music player is designed to facilitate straightforward interaction with music files, providing core features that include adding songs to a playlist, and controlling playback with options to play, pause, resume, restart, stop, and adjust the volume of the music.

The application is structured around three main classes: \`Song\`, \`Media\`, and \`MusicPlayer\`. The \`Song\` class serves as a base for representing a music track with attributes like title and file path. The \`Media\` class extends \`Song\` to include additional details such as the artist's name and provides a method for playing the music using Pygame's audio functionalities. The \`MusicPlayer\` class encapsulates the core functionalities required to manage and control the playback of music, including adjusting volume and seeking within the track.

The main class, \`PlayMusic\`, serves as the interface for user interaction, presenting a menu-driven interface that allows users to perform various operations on their music files. The program uses error handling to manage user inputs and ensure that only valid operations are executed, enhancing the robustness of the application.

4. Functionalities:

❖ **Load Song/Audio File**:

➢ **pygame.mixer.music.load():** Allows users to add a new song to the player by providing the song's title, artist, and file path. The song is then appended to the playlist for future playback.

➢ Why it's important: There is no default sounds or other audio files to play, the user need to load the existing song from the local file path.

❖ **Play Song/Audio File**:

➢ **pygame.mixer.music.play():** Enables users to select and play a song from the playlist. The song begins playback from the start, and the player updates to reflect the current track.

❖ **Pause Song/Audio File:**

➢ **pygame.mixer.music.pause():** Temporarily pauses the currently playing song. This functionality allows users to halt playback and resume it from the same position later.

❖ **Resume Song/Audio File:**

➢ **pygame.mixer.music.unpause():** Resumes playback of a song that has been paused. This feature allows users to continue listening from where they left off.

❖ **Rewind/Restart Playback**:

➢ **pygame.mixer.music.rewind():** Restarts the currently playing song from the beginning. This is useful for users who want to replay the track.

- ❖ **Forward | Backward**:

- ➢ **pygame.mixer.music.set_pos():** Allows users to skip forward or rewind the song by a specified number of seconds. Users input the time in seconds to jump to a particular position in the track.

- ❖ **Set Volume:**

- ➢ **pygame.mixer.music.set_volume():** Adjusts the volume of the playback. Users can set the volume level between 0.0 (mute) and 1.0 (full volume), providing control over the audio output.

These functionalities combined make the MusicPlayer an effective tool for enhancing audio files and keeping user friendly.

5. Input Versatility with Error Handling and Exception Handling:

Input Versatility : The project handles user inputs through a menu-driven interface, allowing for a variety of actions based on user choices. To ensure the application remains robust, input validation is employed to manage different types of user input. For instance, when setting the volume or seeking within a song, the input is validated to ensure it falls within acceptable ranges. If invalid input is provided, such as non-numeric values for volume or time, the application catches these errors using try-except blocks, informs the user of the issue, and prompts for a correct input. This approach enhances the user experience by preventing crashes and ensuring that only valid operations are performed.

Error Handling: In simple words means making sure the system doesn't crash or break when something unexpected happens. It checks the information users give and makes sure it's correct before moving forward.

For example:

- ✓ If a user enters a String format that’s not between 0.0 and 1.0, the system will show an error and ask the user to try again.
- ✓ If a user enters a Invalid format (String and other operators) that’s not Integer value, the system will show an error and ask the user to try again.

The project handles different types of inputs, including song titles, artist names, file paths, and volume levels. It also handles errors and exceptions gracefully by providing user-friendly error messages and prompts. For example, if a user enters an invalid file path, the application will display an error message and prompt the user to enter a valid file path.

This way, the program catches mistakes and guides users to correct them, making the system run smoothly and without issues.

6. Code Implementation:

The PlayMusic class integrates user interaction by presenting a menu and processing user choices. It uses the MusicPlayer instance to perform various operations based on user inputs. The code is structured to handle potential exceptions and errors, ensuring that invalid inputs are managed gracefully.

❖ Song class:

```
        import pygame

class Song():
    def __init__(self, title, file_path):
        self.title = title
        self.file_path = file_path
```

❖ Media Class:

```
class Media(Song):
    def __init__(self, title, artist, file_path):
        super().__init__(title, file_path)
        self.artist = artist

    def play(self):
        pygame.mixer.music.load(self.file_path)
        pygame.mixer.music.play()
```

❖ MusicPlayer Class:

```
class MusicPlayer:
    def __init__(self):
        self.song_list = []
        self.current_song = None
        self.volume = 0.5
        pygame.init()
    def add_song(self, song):
        self.song_list.append(song)

    def play_song(self, song):
        self.current_song = song
        song.play()

    def pause_song(self):
        pygame.mixer.music.pause()

    def resume_song(self):
        pygame.mixer.music.unpause()

    def restart_song(self):
        pygame.mixer.music.rewind()

    def set_pos(self,time):
        self.time=time
        pygame.mixer.music.set_pos(time)

    def stop_song(self):
        pygame.mixer.music.stop

    def set_volume(self, volume):
        self.volume = volume
        pygame.mixer.music.set_volume(volume)
```

- **Play_song Class:**

```
class PlayMusic(MusicPlayer,Media):

    player = MusicPlayer()

    while True:
        print("\nMusic Player Menu:")
        print("1. Add Song")
        print("2. Play Song")
        print("3. Pause Song")
        print("4. Resume Song")
        print("5. Restart Song")
        print("6. Forward | Backward")
        print("7. Stop Song")
        print("8. Set Volume")
        print("9. Exit")

        choice = input("Enter your choice: ")

        if choice == "1":
            title = input("Enter song title: ")
            artist = input("Enter song artist: ")
            file_path = input("Enter song file path: ")
            song = Media(title, artist, file_path)
            player.add_song(song)
            print()
            print(f"Song '{title}' is added and ready to Play.")
        elif choice == "2":
            song_title = input("Enter Song title to play: ")
            for i in player.song_list:
                if i.title == song_title:
                    player.play_song(song)
                    print()
                    print(f"Now Playing...'{song_title}.mp3' composed by '{artist}'")
                    break
            else:
                print("Media not found!")
        elif choice == "3":
            player.pause_song()
            print()
            print(f"'{song_title}.mp3' is Paused.")
        elif choice == "4":
            player.resume_song()
            print()
            print(f"Now Playing...'{song_title}.mp3' composed by '{artist}'")
        elif choice == "5":
            player.restart_song()
            print(f"'{song_title}.mp3' is restarted.")
        elif choice == "6:
            try:
                time=int(input("Set time in seconds to skip:"))
            except ValueError:
```

```
                print("Please Enter in Valid Format.")
                continue
            player.set_pos(time)
            print()
            print(f"Playback Skipped to {time} seconds.")
        elif choice == "7":
            player.stop_song()
            print()
            print(f"'{song_title}.mp3' stopped playing.")
        elif choice == "8":
            try:
                volume = float(input("Enter volume level (0.0 - 1.0): "))
            except ValueError:
                print("Please Enter in Valid Format.")
                continue
            player.set_volume(volume)
            print()
            print(f"Volume is set to {volume}")
        elif choice == "9":
            print("Music Player Closed!...")
            break # breaks the loop
        else: # if the given input not between [ 1 - 9 ]
            print("Invalid choice!")

cheran=PlayMusic()
```

7. Results and Outcomes:

The implementation of the music player successfully provides a functional application with essential playback controls. Users can interact with their music files through a simple menu, add songs, and control playback with ease. The project demonstrates how Python and Pygame can be leveraged to create a basic but effective media player. The key outcomes include a user-friendly interface and reliable audio playback, with all core functionalities working as intended.

8. Conclusion:

In conclusion, the music player project achieves its goal of providing a straightforward and efficient tool for managing and playing music files. By focusing on core functionalities and leveraging the Pygame library, the project delivers a practical solution for users seeking a lightweight music player. Future developments could include additional features such as playlist management, advanced audio effects, or integration with online music services to enhance the application's capabilities.