**Use Case-1**

**Music Player Application**

This use case demonstrates the usage of the MusicPlayer class to create a music player application. The application allows users to manage their playlist, play, pause, resume, stop songs, and add new songs to the playlist.

**Use Case Steps:**

* The user launches the music player application.
* The application initializes the MusicPlayer class with an existing playlist.
* The application presents the user with a menu of actions to choose from.
* The user selects an action:
* Play a Song: The user provides the song number from the playlist. The application checks if the song exists in the playlist and plays it if found.
* Pause the Current Song: The application pauses the currently playing song, if any.
* Resume the Current Song: The application resumes the currently paused song, if any.
* Stop the Current Song: The application stops the currently playing or paused song, if any.
* Add a Song to the Playlist: The user provides the name of the song to add. The application checks if the song already exists in the playlist and adds it if not.
* Exit: The user exits the music player application.
* The application performs the selected action and displays the appropriate message or performs the corresponding functionality.
* The application loops back to the menu, allowing the user to select another action.
* The application continues to process user actions until the user chooses to exit.

**Problem Statement: Music Player Application**

You have been tasked with developing a music player application that allows users to manage their playlist and control the playback of songs. The application should provide a user-friendly interface with various options to play, pause, resume, stop songs, and add new songs to the playlist.

**Requirements:**

1. The application should implement a MusicPlayer class that encapsulates the functionality of the music player.
2. The MusicPlayer class should have the following attributes:

* playlist: A list containing the names of songs in the playlist.
* current\_song: A string representing the name of the currently playing song.

1. The MusicPlayer class should have the following methods:

* play(song): Plays the specified song if it exists in the playlist.
* pause(): Pauses the currently playing song, if any.
* resume(): Resumes the currently paused song, if any.
* stop(): Stops the currently playing or paused song, if any.
* add\_song(song): Adds a new song to the playlist if it is not already present.
* get\_playlist(): Returns the current playlist.

1. The application should allow the user to interact with the music player through a menu-driven interface.
2. The menu should provide the following options:

* Play a Song: Allows the user to enter the song number from the playlist to play.
* Pause the Current Song: Pauses the currently playing song.
* Resume the Current Song: Resumes the currently paused song.
* Stop the Current Song: Stops the currently playing or paused song.
* Add a Song to the Playlist: Allows the user to enter the name of a new song to add to the playlist.
* Exit: Terminates the application.

1. The application should validate user input and provide appropriate error messages for invalid choices or inputs.

**Constraints:**

* The playlist can contain any number of songs.
* Song names are unique within the playlist.
* The application should handle cases where the user tries to perform actions on a non-existent song or when no song is currently playing or paused.

**AI implementation**

In this AI-generated implementation, the code defines a MusicPlayer class that represents a simple music player. It has methods to play a song, pause the current song, resume a paused song, stop the current song, and add a new song to the playlist.

The code creates an instance of the MusicPlayer class with an initial playlist. Then, it enters a while loop that presents a menu of options to the user and performs the selected action based on the user's input.

1. Intelligent Decision-Making: The program allows users to add books, display books by author, calculate the total number of pages, display a table of books, and remove books. These actions

involve decision-making based on user input and data manipulation. Although the program does not employ advanced AI techniques, it demonstrates basic decision-making capabilities.

2. Data Analysis: The program maintains a list of books and their details, such as titles, authors, and page counts. AI algorithms could be employed to analyze this data, identify patterns, and generate insights. For instance, data analysis techniques could help identify popular authors, genres, or trends in reading habits.

3.Natural Language Processing (NLP): Use NLP techniques to enhance search capabilities, allowing users to search for bank accounts using natural language queries.

**THE CODE OF MusicPlayer SYSTEM**

class MusicPlayer:

def \_\_init\_\_(self, playlist):

self.playlist = playlist

self.current\_song = ''

def play(self, song):

if song in self.playlist:

self.current\_song = song

print("Now playing: " + song)

else:

print(song + " is not in the playlist.")

def pause(self):

if self.current\_song:

print("Paused: " + self.current\_song)

else:

print("No song is currently playing.")

def resume(self):

if self.current\_song:

print("Resuming: " + self.current\_song)

else:

print("No song is currently paused.")

def stop(self):

if self.current\_song:

print("Stopped: " + self.current\_song)

self.current\_song = ''

else:

print("No song is currently playing.")

def add\_song(self, song):

if song not in self.playlist:

self.playlist.append(song)

print("Added " + song + " to the playlist.")

else:

print(song + " is already in the playlist.")

# Create an instance of MusicPlayer

playlist = ['song1.mp3', 'song2.mp3', 'song3.mp3']

player = MusicPlayer(playlist)

while True:

print("Choose an action:")

print("1. Play a song")

print("2. Pause the current song")

print("3. Resume the current song")

print("4. Stop the current song")

print("5. Add a song to the playlist")

print("0. Exit")

choice = input("Enter your choice: ")

if choice == "1":

song\_index = int(input("Enter the song number to play: ")) - 1

if song\_index >= 0 and song\_index < len(playlist):

player.play(playlist[song\_index])

else:

print("Invalid song number.")

elif choice == "2":

player.pause()

elif choice == "3":

player.resume()

elif choice == "4":

player.stop()

elif choice == "5":

new\_song = input("Enter the name of the song to add: ")

player.add\_song(new\_song)

elif choice == "0":

break

else:

print("Invalid choice. Please try again.\n")

**Explaning of the code:**

1. The MusicPlayer class is defined with an \_\_init\_\_ method, which initializes the player with a playlist and sets the current\_song to an empty string.
2. The play method takes a song as input. If the song is present in the playlist, it sets the current\_song to the input song and prints a message indicating that the song is being played. Otherwise, it prints a message stating that the song is not in the playlist.
3. The pause method checks if there is a current song playing. If so, it prints a message indicating that the current song has been paused. If no song is currently playing, it prints a message stating that there is no song playing.
4. The resume method checks if there is a current song that has been paused. If so, it prints a message indicating that the current song has been resumed. If no song is currently paused, it prints a message stating that there is no song paused.
5. The stop method checks if there is a current song playing. If so, it prints a message indicating that the current song has been stopped and sets the current\_song to an empty string. If no song is currently playing, it prints a message stating that there is no song playing.
6. The add\_song method takes a song as input. If the song is not already in the playlist, it appends the song to the playlist and prints a message indicating that the song has been added. If the song is already in the playlist, it prints a message stating that the song is already in the playlist.
7. An instance of the MusicPlayer class is created with a predefined playlist.
8. The code enters an infinite loop using while True. It displays a menu of options for the user to choose from.
9. The user's choice is captured through the input function and stored in the choice variable.
10. Based on the user's choice, the corresponding action is performed:

* If the choice is "1", the user is prompted to enter the song number to play. If the input is a valid song number within the range of the playlist, the play method is called with the corresponding song from the playlist.
* If the choice is "2", the pause method is called.
* If the choice is "3", the resume method is called.
* If the choice is "4", the stop method is called.
* If the choice is "5", the user is prompted to enter the name of the song to add, and the add\_song method is called with the input song.
* If the choice is "0", the loop is terminated, and the program exits.
* If the choice is not a valid option, an error message is printed.
* After each action, the loop repeats, displaying the menu again for the user to choose another action.

**Output of LIBRARY MANAGEMENT code**

Choose an action:

1. Play a song

2. Pause the current song

3. Resume the current song

4. Stop the current song

5. Add a song to the playlist

0. Exit

Enter your choice: 1

Enter the song number to play: 1

Now playing: song1.mp3

Choose an action:

1. Play a song

2. Pause the current song

3. Resume the current song

4. Stop the current song

5. Add a song to the playlist

0. Exit

Enter your choice: 2

Paused: song1.mp3

Choose an action:

1. Play a song

2. Pause the current song

3. Resume the current song

4. Stop the current song

5. Add a song to the playlist

0. Exit

Enter your choice: 3

Resuming: song1.mp3

Choose an action:

1. Play a song

2. Pause the current song

3. Resume the current song

4. Stop the current song

5. Add a song to the playlist

0. Exit

Enter your choice: 4

Stopped: song1.mp3

Choose an action:

1. Play a song

2. Pause the current song

3. Resume the current song

4. Stop the current song

5. Add a song to the playlist

0. Exit

Enter your choice: 5

Enter the name of the song to add: song5.mp3

Added song5.mp3 to the playlist.

Choose an action:

1. Play a song

2. Pause the current song

3. Resume the current song

4. Stop the current song

5. Add a song to the playlist

0. Exit

Enter your choice: 0