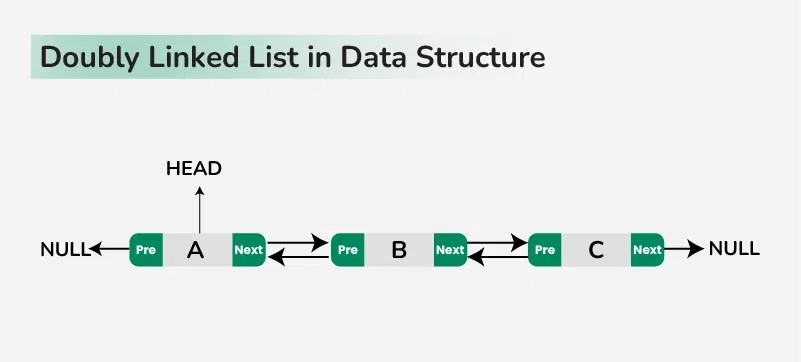
**INTRODUCTION**

* 1. **Doubly Linked List**

A doubly linked list is a fundamental linear data structure comprising nodes, each containing a data field alongside two pointers: one referencing the next node and another pointing to the previous node. This bidirectional linkage enables traversal in both forward and backward directions, rendering it highly suitable for applications necessitating sequential access and efficient insertion and deletion operations.

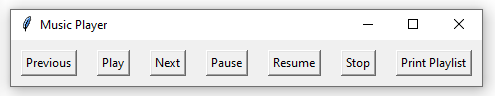
In our project, the choice of a doubly linked list as the underlying data structure is deliberate and strategic. Its versatility aligns seamlessly with the requirements of our music application, allowing for dynamic playlist management and effortless navigation through music collections. By leveraging the bidirectional traversal capability of a doubly linked list, users can seamlessly skip tracks, play music, append to playlists, and navigate through their music libraries with ease. Notably, the ability to traverse in both forward and backward directions enhances user experience and functionality, while the efficiency of delete operations with a given node pointer streamlines playlist management. Additionally, the quick insertion of new nodes before a given node offers flexibility in playlist customization.

A diagram of a single linked list

Description automatically generatedIn the context of a music app, each song in the playlist can be represented as a node in the doubly linked list. This structure enables easy navigation through the playlist, supporting features like moving to the next or previous song with minimal complexity. Additionally, the ability to traverse the playlist in reverse order enhances user experience by allowing seamless backward navigation.

* 1. **Music App using Doubly Linked List**

The Music Player application we are developing utilizes doubly linked lists to efficiently manage playlists and tracks. Built with Python and leveraging Pygame for audio playback and Tkinter for the graphical interface, our project aims to offer users a versatile platform for organizing and enjoying their music collections. This report delves into the design, implementation, and functionalities of our Music Player application, exploring its features, architecture, and future prospects in enhancing the user experience of listening to music in a digital environment.



**APPLICATION**

The Music Player project developed using Python, Pygame, and Tkinter, offers a versatile platform for organizing and playing music collections. In this section, we explore various applications of the music player project in different contexts, highlighting its potential uses and benefits.

**3.1 Personal Music Player**

The music player application can be deployed as a standalone music player for personal use, allowing users to create and manage their playlists, navigate through tracks, and enjoy their favourite music seamlessly.

* 1. **Music Education**

Educational institutions such as schools and music academies can utilize the music player application to facilitate music education programs. Students can explore different genres, learn about music composition, and practice musical instruments with interactive playlists.

* 1. **Meditation and Relaxation**

The music player can also serve as a meditation and relaxation tool, offering curated playlists of soothing music and ambient sounds. Users can create calming playlists to unwind, reduce stress, and promote relaxation.

* 1. **Multimedia and Presentation**

Teachers and educators can incorporate the music player into multimedia presentations to enhance learning experiences. By integrating audio tracks into presentations, educators can create engaging and immersive learning materials for students.

* 1. **Advantages**
     1. **User-Friendly Interface**

The graphical user interface (GUI) developed using Tkinter offers an intuitive layout with easy-to-use controls. This enhances user experience and makes the application accessible to users of all levels of technical proficiency.

* + 1. **Playback Control**

The application provides seamless playback control with features like play, pause, resume, stop, and track navigation. Users can control playback easily, allowing them to enjoy uninterrupted music playback.

* + 1. **Minimal Dependencies**

The project has minimal dependencies, primarily relying on Python's standard library, Pygame for audio playback, and Tkinter for GUI development. This reduces the complexity of setup and deployment, making the application easy to install and use.

* + 1. **Project-Based Learning**

The development of the music player project offers valuable opportunities for project-based learning. Developers can gain practical experience in software development, project management, version control, and collaboration, honing their skills and building a portfolio of work.

**OPERATION**

The operation section of the Music Player project outlines the key functionalities provided by the application to users for managing and controlling their music playback experience. These operations encompass essential actions such as navigating between songs, initiating playback, pausing/resuming playback, stopping playback, and viewing playlist details. Each operation serves a specific purpose in facilitating seamless interaction with the music player interface and enhancing the user's ability to organize, navigate, and enjoy their music collection.

* 1. **Previous:**

The "Previous" operation enables users to navigate to the previous song in the playlist. When activated, the application moves the current song pointer to the preceding node in the playlist, facilitating seamless navigation between songs during playback.

* 1. **Play:**

The "Play" operation initiates playback of the currently selected song in the playlist. Upon invocation, the application loads the audio file of the selected song and begins playback, providing users with immediate access to their music collection.

* 1. **Next:**

"Next" allows users to move to the next song in the playlist. When triggered, the application advances the current song pointer to the subsequent node in the playlist and starts playback of the newly selected song, ensuring continuous music enjoyment.

* 1. **Pause:**

The "Pause" operation suspends playback of the currently playing song. Upon activation, the application temporarily halts playback, giving users the flexibility to pause and resume playback at their convenience without losing track of their listening progress.

* 1. **Resume:**

"Resume" restarts playback of the currently paused song. Upon invocation, the application resumes playback from the paused position, seamlessly continuing the music experience without interruption, enhancing user control and convenience.

* 1. **Stop:**

The "Stop" operation halts playback of the currently playing song and resets playback to the beginning of the track. When activated, the application stops playback, providing users with a convenient way to end the current listening session and prepare for a new selection.

* 1. **Print Playlist:**

"Print Playlist" displays the contents of the playlist, including song titles and artists, to the user interface or console output. This operation provides users with visibility into their playlist contents, facilitating organization and management of their music collection.

A screenshot of a computer

Description automatically generated

**IMPLEMENTATION**

The code defines two classes: "Song" and "Playlist". The "Song" class represents individual songs with attributes such as title, artist, and file path. Each song object also has references to the next and previous songs in the playlist, forming a linked list structure. The "Playlist" class manages a collection of songs and provides operations for adding songs, playing, pausing, and navigating through the playlist. It maintains references to the head and tail of the playlist, as well as the currently playing song and its playback status. The code establishes a foundation for building a music player application with features for managing playlists and controlling audio playback.

* 1. **Song Class**

class Song:

def \_\_init\_\_(self, title, artist, file\_path):

self.title = title

self.artist = artist

self.file\_path = file\_path

self.next = None

self.prev = None

* 1. **Playlist Class**

class Playlist:

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

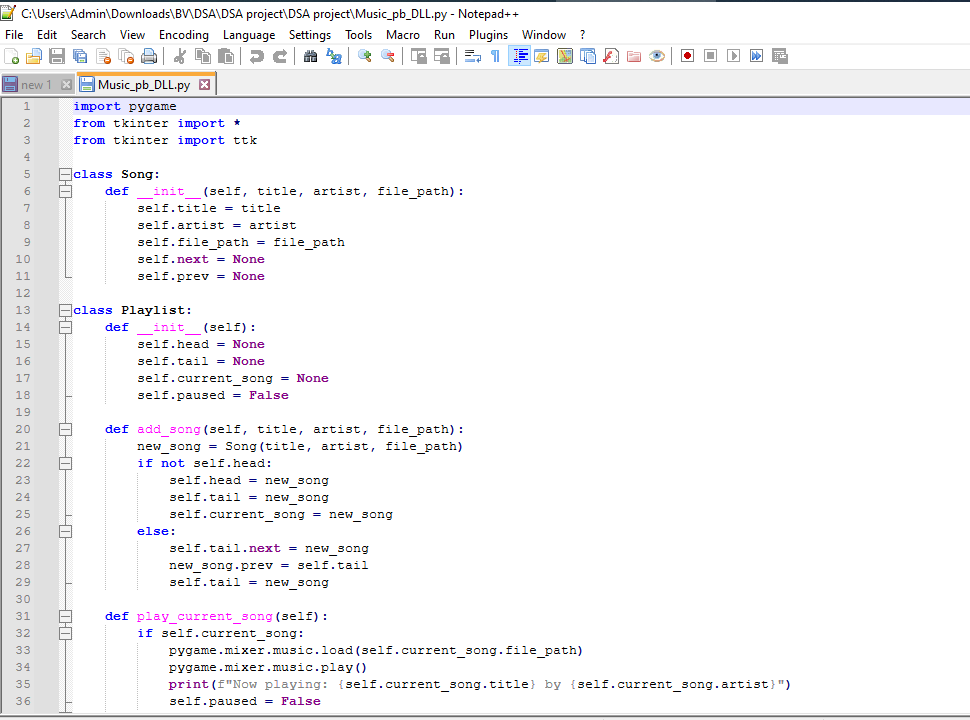
self.head = None

self.tail = None

self.current\_song = None

self.paused = False

* 1. **Code Snippets**

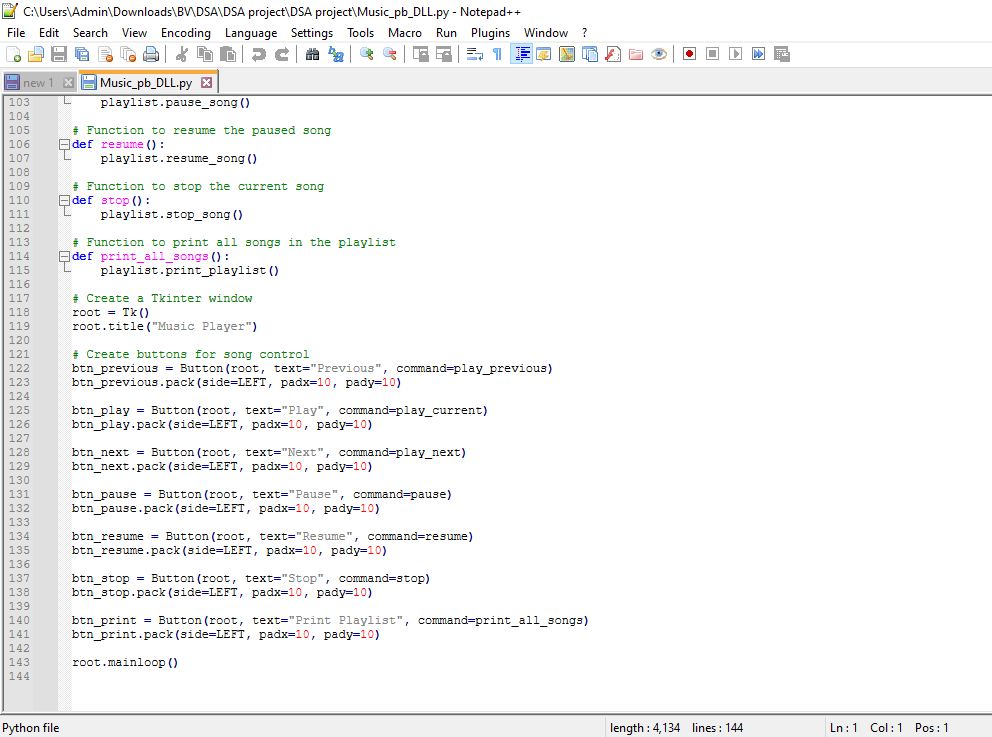


A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

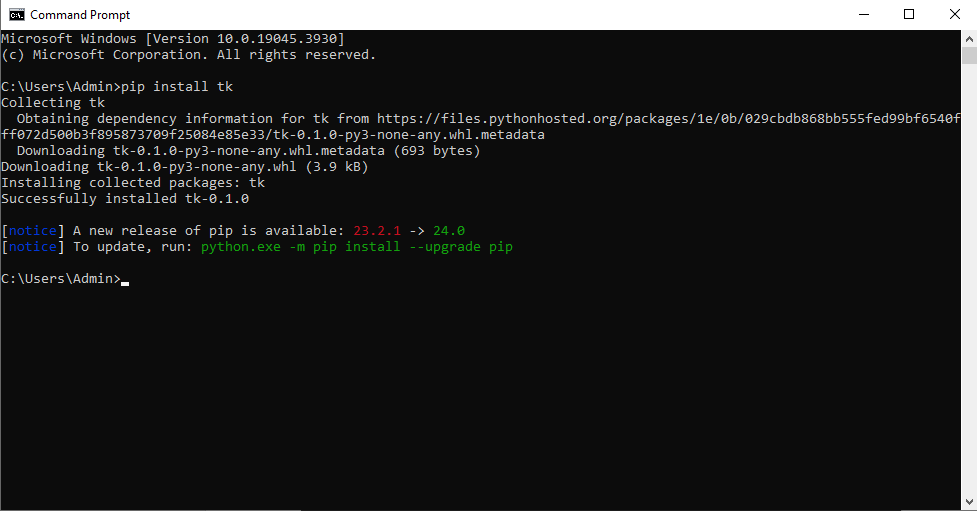


* 1. **Tools and Api’s Used**
     1. **Pygame**

A computer screen shot of a program

Description automatically generatedPygame is a set of Python modules designed for writing video games and multimedia applications. It is used in the project for audio playback functionality, allowing the music player to load and play audio files.

* + 1. **Tkinter**

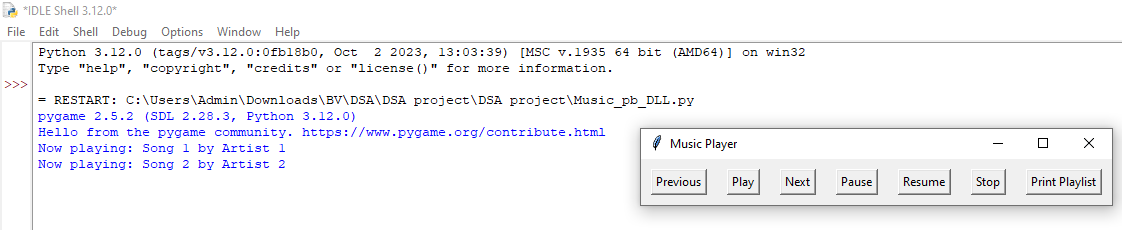
Tkinter is Python's standard GUI (Graphical User Interface) toolkit. It is utilized for creating the graphical interface of the music player application, including buttons for controlling playback and displaying playlist information.

**OUTPUTS**

When executed the output include Messages displayed in the console indicating the status of the music player operations. Interaction with the graphical user interface (GUI) created using Tkinter includes buttons for controlling playback, displaying playlist information, and responding to user actions, and audio playback through the system’s audio output device.

output in console when song is played


output in console when play button is clicked.

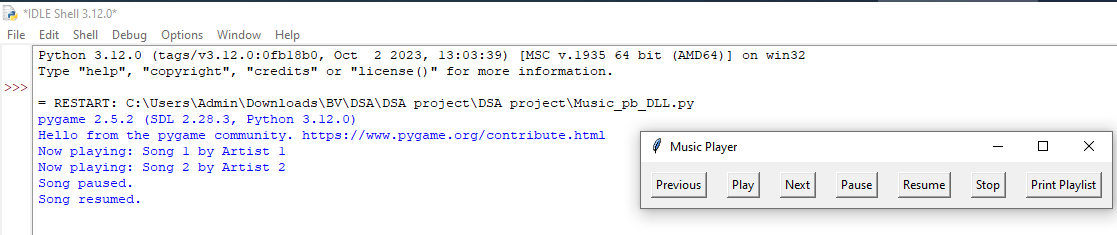


output in console when next button is clicked.

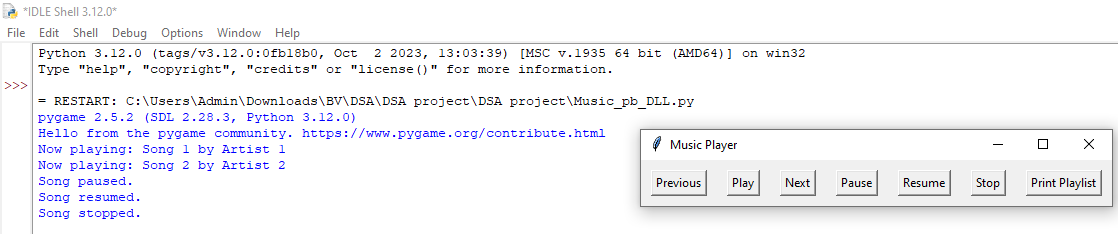
A screenshot of a computer

Description automatically generated

output when pause button is clicked.



output in console when resume button is clicked.



output when the stop button is clicked.

A screenshot of a computer

Description automatically generated

output when the print playlist button is clicked.

**CONCLUSION**

The Music Player project represents a significant achievement in providing users with an accessible tool for managing their music collections. By utilizing the capabilities of Python, Pygame, and Tkinter, the application offers a seamless interface for organizing playlists, navigating through tracks, and controlling playback. With features such as playlist management and dynamic playback control, users can tailor their music listening experience to their preferences with ease. Looking forward, the project holds great potential for future enhancements that could further elevate its functionality and usability. One promising avenue is the integration of advanced search and filtering options, allowing users to quickly locate specific songs or albums within their playlists. Additionally, the implementation of customizable themes and skins would provide users with the ability to personalize the application's appearance to suit their tastes. Furthermore, the project could benefit from the integration of online streaming services, enabling users to access a wider range of music content directly within the application. Extending compatibility to other platforms, such as mobile devices, would also expand the reach of the Music Player, making it accessible to a broader audience of users.

**REFERENCES**

1. <https://docs.python.org/3/installing/index.html>
2. <https://www.geeksforgeeks.org/introduction-to-pygame/>
3. <https://pypi.org/project/pygame/>
4. <https://docs.python.org/3/library/tkinter.html>
5. <https://stackoverflow.com/questions/20021457/playing-mp3-song-on-python>
6. <https://www.youtube.com/watch?v=PSFM8-byvAE>
7. <https://www.youtube.com/watch?v=DGeDcxul5Zk>
8. <https://stackoverflow.com/questions/68698823/why-is-a-doubly-linked-list-used-in-a-music-player-with-next-and-previous-bu>
9. <https://medium.com/journey-of-one-thousand-apps/data-structures-in-the-real-world-508f5968545a>