

JUKEBOX USING PYTHON

INTERDISCIPLINARY PROJECT

Submitted in partial fulfillment of the
requirements for the award of
Bachelor of Technology degree in Information Technology

By

AMAY KANCHHIT (41120016)



DEPARTMENT OF INFORMATION TECHNOLOGY

SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Category-1 University by UGC

Accredited with Grade “A++” by NAAC

12B Status by UGC |Approved by AICTE

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600119

MAY - 2024



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited "A++" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF INFORMATION TECHNOLOGY

BONAFIDE CERTIFICATE

This is to certify that this Professional Training is the bonafide work of **MR. AMAY KANCHHIT (41120016)** who carried out the project entitled "**JUKEBOX USING PYTHON**" under my supervision from **January 2024** to **May 2024**.

Internal Guide
Dr. S. URMELA, M.Tech., Ph.D.,

Head of the Department
Dr. R. SUBHASHINI, M.E., Ph.D.,

Submitted for Viva voice Examination held on 08-05-2024

Internal Examiner

External Examiner

DECLARATION

I AMAY KANCHHIT hereby declare that the Project Report entitled "JUKEBOX" done by me under the guidance of **Dr. S. URMELA, M.Tech., Ph.D.**, is submitted in partial fulfillment of the requirements for the award of Bachelor of Technology degree in Information Technology.

DATE: 08/05/2024

PLACE: Chennai

Amay Kanchhit

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr.T.SASIKALA M.E., Ph.D., Dean**, School of Computing **Dr.R.SUBHASHINI M.E., Ph.D.**, Head of the Department of Information Technology for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. S. URMELA, M.Tech., Ph.D.**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Information Technology** who were helpful in many ways for the completion of the project.

CERTIFICATE OF COMPLETION



Elite

NPTEL Online Certification

(Funded by the MoE, Govt. of India)



This certificate is awarded to

AMAY KANCHHIT

for successfully completing the course

Programming, Data Structures and Algorithms using Python

with a consolidated score of **67** %

Online Assignments	25/25	Proctored Exam	42.31/75
--------------------	-------	----------------	----------

Total number of candidates certified in this course: **2042**

Devendra Jalihal

Prof. Devendra Jalihal
Chairperson,
Centre for Outreach and Digital Education, IITM

Jan-Mar 2024

(8 week course)

Prof. Andrew Thangaraj

Prof. Andrew Thangaraj
NPTEL, Coordinator
IIT Madras



Indian Institute of Technology Madras



Roll No: NPTEL24CS45S244105914

To verify the certificate



No. of credits recommended: 2 or 3

ABSTRACT

JukeBox a python-based basic Music Player Project. In an era dominated by digital streaming services, the charm of classic jukeboxes remains timeless. This project, titled "Jukebox," aims to recreate the nostalgic experience of selecting and playing music in a digital format using Python programming language. The Jukebox project utilizes Python's versatile libraries, including tkinter for the graphical user interface (GUI) and pygame for audio playback. Users interact with the Jukebox through a simple, intuitive interface, reminiscent of traditional jukebox designs. They can browse through a library of preloaded songs or add their own music files to create a personalized playlist. Key features of the Jukebox include song selection, play, pause, skip, and volume control functionalities. Additionally, the project incorporates basic search and sorting algorithms to enhance the user experience, allowing for efficient navigation through the music library. The implementation of Jukebox not only serves as a practical application of Python programming concepts but also offers an opportunity for users to engage with music in a nostalgic and interactive manner. This project serves as a foundation for further exploration and customization, inviting users to expand its functionality and design according to their preferences.

TABLE OF CONTENTS

Ch No.	TITLE	Page No.
	ABSTRACT	vi
	LIST OF ABBREVIATIONS	ix
	LIST OF FIGURES	x
1	INTRODUCTION	1
1.1	Overview of Music Player	2
1.2	Technologies Used	4
1.3	Programming Language	6
1.4	Frameworks or Libraries	7
1.5	Objectives	8
2	METHODOLOGY	10
2.1	Initial Project Requirements	10
2.2	Proposed System Architecture	13
2.3	Flow Diagram	17
2.4	Modules	20
2.5	Development and Deployment Setup	23

3	RESULTS AND DISCUSSION	26
4	CONCLUSION AND FUTURE WORK	29
	4.1 Conclusion	29
	4.2 Future Work	29
	REFERENCES	31
	APPENDICES	32
	A. Sample Code	32
	B. Screen Shots	35

LIST OF ABBREVIATIONS

ABBREVIATION		EXPANSION
UI	-	User Interface
UX	-	User Experience
IOS	-	iPhone Operating System
UAT	-	User Acceptance Testing
IDE	-	Integrated Development Environment
CLI	-	Command Line Interface
MP	-	Music Player

LIST OF FIGURES

Figure No.	Name of the Figure	Page No.
1.1	Music Player Interface	05
2.1	Sequence Diagram	19
2.2	Flowchart for Playback Control	25
3.1	Project Architecture Diagram	28

CHAPTER 1

INTRODUCTION

Introducing "JUKEBOX" - a Python-powered music player that seamlessly blends functionality with simplicity, promising a harmonious listening experience like no other. Crafted with precision and passion, JUKEBOX is designed to elevate your music journey to new heights.

Imagine a world where your music collection is not just a list of songs, but an immersive sonic landscape waiting to be explored. With JUKEBOX, that vision becomes a reality. Whether you're a casual listener or a die-hard audiophile, our player caters to your every need, offering a sleek interface and powerful features that make navigating and enjoying your music effortless.

Featuring a robust backend written in Python, JUKEBOX is as efficient as it is elegant. Its intuitive design ensures that even novice users can dive right in and start enjoying their favorite tunes within moments. But don't let its simplicity fool you - under the hood, JUKEBOX boasts a wealth of advanced functionalities, from customizable playlists to seamless integration with your existing music library. Say goodbye to clunky interfaces and cumbersome navigation - JUKEBOX puts the power of music at your fingertips, allowing you to rediscover your favorite tracks with ease. Whether you're unwinding after a long day or hosting a party with friends, let JUKEBOX be your soundtrack to every moment.

Join us as we embark on a musical journey like no other. Welcome to JUKEBOX - where the rhythm of life meets the melody of technology.

1.1. OVERVIEW OF MUSIC PLAYER

The JUKEBOX Music Player in Python is a software application designed to provide users with a seamless and intuitive experience for organizing, browsing, and playing their music library. Built using Python programming language along with relevant libraries such as Tkinter for the graphical user interface (GUI) and Pygame for audio playback, this project offers a versatile solution for managing music collections.

Key Features:

User-friendly Interface: The music player boasts an easy-to-navigate interface, allowing users to quickly access their music library, playlists, and playback controls.

Library Management: Users can import their music library into the application, organize tracks by artist, album, genre, or other metadata, and easily search for specific songs.

Playlist Creation: The music player enables users to create custom playlists based on their preferences, allowing them to curate their favorite tracks for seamless playback.

Audio Playback: Leveraging the Pygame library, the application provides smooth and reliable audio playback with support for various audio formats, including MP3, WAV, FLAC, and more.

Playback Controls: Users have access to essential playback controls such as play, pause, stop, skip, and volume adjustment, offering flexibility and convenience during music playback.

Visualizations: For an enhanced listening experience, the music player may include visualizations such as spectrum analyzers or waveform displays, providing users with captivating visuals synchronized to the music.

Customization Options: Users can customize the appearance and behavior of the music player according to their preferences, including theme selection, playback settings, and hotkey assignments.

Cross-platform Compatibility: Being developed in Python, the music player project is inherently cross-platform, ensuring compatibility with major operating systems such as Windows, macOS, and Linux.

Future Enhancements Online Streaming: Integration with online music streaming services to expand the library beyond locally stored tracks.

Lyrics Display: Incorporating a feature to display synchronized lyrics for the currently playing song.

Smart Recommendations: Implementing machine learning algorithms to offer personalized music recommendations based on listening history and preferences.

Social Sharing: Adding functionality for users to share their favorite songs or playlists with friends on social media platforms.

Mobile App Integration: Developing companion mobile applications for seamless synchronization and remote control of the music player.

The Music Player Project in Python offers a robust solution for music enthusiasts to manage and enjoy their music collections effectively. With its user-friendly interface, comprehensive features, and potential for future enhancements, this project serves as a versatile platform for music playback and organization.

1.2. TECHNOLOGIES USED

For a music player project named "JUKEBOX" developed using Python, I used a combination of libraries and frameworks to achieve various functionalities. Here's a list of technologies commonly used in this project:

Python: The core programming language used for developing the application logic and functionalities. Tkinter or PyQt: GUI (Graphical User Interface) frameworks in Python for creating the user interface of the music player. Tkinter is Python's standard GUI (Graphical User Interface) toolkit, while PyQt is another popular option for creating desktop applications with Python.

pygame: A cross-platform set of Python modules designed for writing video games. It can be utilized for audio playback functionalities in the music player.

sqlite3: Python's built-in SQLite database library, used for managing and storing metadata related to songs such as artist name, album title, genre, etc.

Mutagen: A Python module to handle audio metadata. It can be used to extract metadata from audio files, such as MP3s, and display them in the music player interface.

Requests: If the music player incorporates online functionalities like streaming music, the Requests library can be used for making HTTP requests to fetch data from online sources.

Beautiful Soup: If you're scraping online sources for music information or album art, Beautiful Soup can be handy for parsing HTML and XML documents. FFmpeg: A multimedia framework that can be used to decode, encode, transcode, mux, demux, stream,

filter, and play almost anything that humans and machines have created. It can be integrated into the music player for advanced audio processing functionalities.

PyDub: A high-level audio processing library in Python. It can be used for tasks like audio playback, manipulation, and conversion.

Spotipy: If your music player integrates with Spotify, Spotipy is a Python library for the Spotify Web API. It can be used to fetch user playlists, track information, and more.

These technologies are used to create "JUKEBOX" music player application in Python.

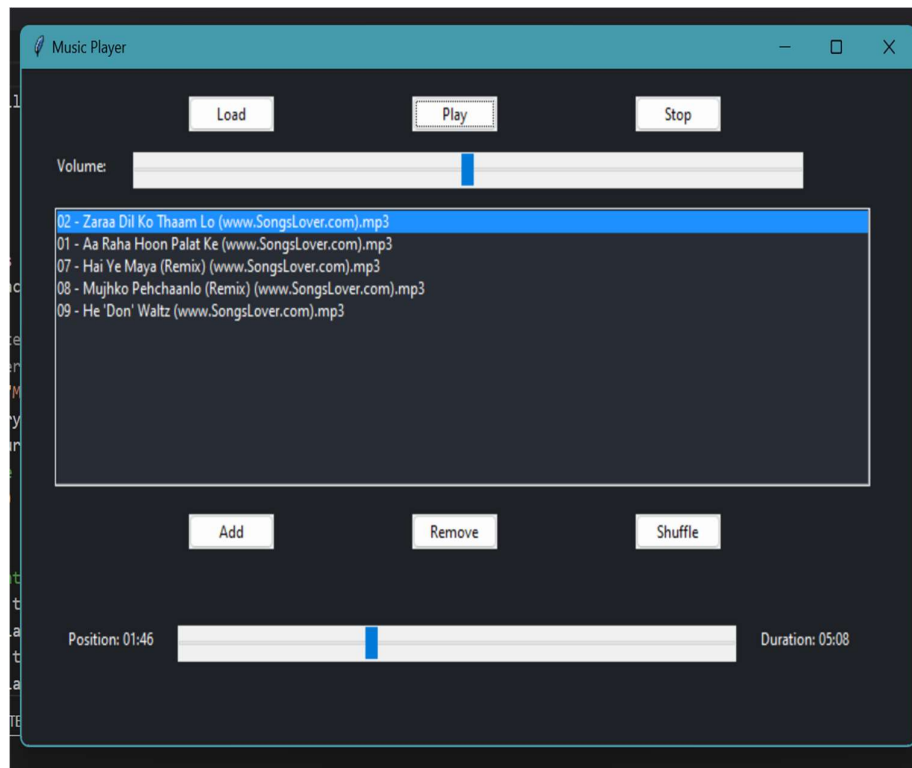


Fig no. 1.1 Music Player Interface

1.3. PROGRAMMING LANGUAGE

Python: Python serves as the primary programming language for the Jukebox project due to its simplicity, readability, and versatility. Its extensive ecosystem of libraries and frameworks makes it an ideal choice for developing various applications, including music players. Tkinter or PyQt: To design the user interface (UI) for the Jukebox, you can utilize either Tkinter or PyQt, both of which are Python bindings for the popular cross-platform GUI toolkit, Qt. Tkinter is the standard GUI toolkit included with Python, offering a simple way to create basic interfaces. On the other hand, PyQt provides more advanced features and a polished look but requires the PyQt library to be installed separately. PyGame or Pyglet: These libraries are well-suited for handling audio playback in Python. PyGame is a set of Python modules designed for writing video games, but it also includes functionality for playing and managing audio files. Pyglet, on the other hand, is a multimedia library specifically focused on creating games and interactive media applications. Both PyGame and Pyglet offer features for playing various audio formats, managing playlists, and controlling playback. Mutagen: Mutagen is a Python module for handling audio metadata, such as tags in MP3, OGG, FLAC, and other audio file formats. It allows you to read and write metadata tags, extract information about audio files (e.g., artist, title, album), and manipulate audio file properties programmatically. Integrating Mutagen into your Jukebox project enables you to display relevant information about the currently playing track and organize your music library effectively. SQLite: SQLite is a lightweight, serverless relational database management system that is often used in Python applications for storing and managing structured data. You can utilize SQLite to create a database for storing information about your music library, such as track metadata, playlists, and user preferences.

1.4. FRAMEWORKS OR LIBRARIES

PyQt or Tkinter: These are Python libraries for creating graphical user interfaces (GUIs). PyQt is more feature-rich and modern, while Tkinter is lightweight and comes bundled with Python.

I have used these to design the user interface for my JUKEBOX, including buttons for play, pause, skip, volume control, and playlist management.

pygame: pygame is a popular library for multimedia applications, especially for handling audio. It provides functionalities for loading and playing audio files in various formats such as MP3, WAV, etc. You can use pygame to handle the playback of music tracks in your JUKEBOX.

SQLite (optional): If your JUKEBOX needs to manage a library of music tracks and playlists, you might consider using SQLite, which is a lightweight relational database management system.

SQLite can help you store information about music tracks, such as their titles, artists, genres, and file paths, as well as playlist information. This can be useful for implementing features like creating and managing playlists, searching for specific tracks, and maintaining a library of music. Requests (for web-based features): If you want to implement features like online streaming, fetching lyrics, or integrating with online music services, you might use the Requests library to make HTTP requests. Requests can be used to communicate with web APIs to fetch data such as lyrics, album artwork, or even streaming audio.

Mutagen: Mutagen is a Python library for reading and writing metadata in audio files. It can be useful for extracting metadata such as artist name, album, track number, etc., from music files.

This metadata can be displayed in your JUKEBOX's user interface or used for organizing and searching music tracks.

1.5. OBJECTIVES

The objective of the "JUKEBOX" project is to create a versatile and user-friendly music player application using Python along with relevant libraries and frameworks. The primary aim is to provide a seamless music listening experience for users while incorporating various features and functionalities. Below are the key objectives of the JUKEBOX project:

Music Playback: The core functionality of JUKEBOX is to play music files from the user's library. It should support popular audio formats like MP3, WAV, FLAC, etc. The playback mechanism should be smooth and efficient, allowing users to enjoy their favorite tracks without interruption.

User Interface (UI): Designing an intuitive and visually appealing user interface is crucial for enhancing the user experience. The UI should allow users to navigate through their music library effortlessly, browse playlists, and control playback functions such as play, pause, skip, and volume adjustment.

Library Management: Implementing robust library management features is essential for organizing the user's music collection efficiently. Users should be able to add, remove, and edit metadata of songs, create playlists, and categorize their music library based on artists, albums, genres, etc.

Search and Filtering: Incorporating search and filtering capabilities enables users to quickly locate specific songs or albums within their music library. Users should be able to search for songs by title, artist, album, or genre, and apply filters to refine search results.

Customization Options: Providing customization options allows users to personalize their music listening experience according to their preferences. This may include features such as customizable themes, equalizer settings, and playback preferences like repeat and shuffle modes.

Cross-Platform Compatibility: Ensuring cross-platform compatibility enables users to access the JUKEBOX application across different operating systems. By utilizing Python and compatible libraries, the application can be deployed on Windows, macOS, and Linux platforms, reaching a broader audience.

Performance Optimization: Optimizing the performance of the application is crucial for delivering a smooth and responsive user experience. This involves minimizing resource usage, optimizing code execution, and implementing caching mechanisms to enhance loading times and responsiveness.

Error Handling and Stability: Implementing robust error handling mechanisms is essential for detecting and handling errors gracefully. The application should provide informative error messages to users in case of any issues and ensure stability and reliability during music playback.

Integration with External Services: Integrating with external music services or APIs can enhance the functionality of the JUKEBOX application. This may include features such as fetching album artwork, retrieving lyrics, or accessing streaming services to expand the music catalog available to users.

Documentation and Support: Providing comprehensive documentation and user support resources ensures that users can easily install, configure, and use the JUKEBOX application. Clear documentation should cover installation instructions, usage guidelines, troubleshooting tips, and frequently asked questions (FAQs). By achieving these objectives, the JUKEBOX project aims to deliver a feature-rich and user-friendly music player application that caters to the diverse needs of music enthusiasts, providing them with an enjoyable and immersive listening experience.

CHAPTER 2

METHODOLOGY

2.1. INITIAL PROJECT REQUIREMENTS

The JUKEBOX is a music player application developed using Python and various libraries and frameworks. The aim of the project is to create a user-friendly and efficient music player with features that cater to music enthusiasts' needs.

Initial Project Requirements: User Interface (UI): The UI should be intuitive, visually appealing, and easy to navigate. It should include features such as play, pause, skip, volume control, shuffle, repeat, and a progress bar. The ability to create and manage playlists should be incorporated. Support for multiple themes and customizable skins would enhance user experience.

Music Library Management: The application should allow users to import their music library from their local storage. Support for various file formats like MP3, WAV, FLAC, etc., should be included. The ability to automatically fetch and display metadata (e.g., artist, album, genre) of the imported songs would be beneficial. Sorting and searching functionalities to easily locate songs within the library.

Playback Features: Smooth playback with support for play, pause, stop, skip forward/backward, and volume control. Implement a feature to create and manage playlists. Support for shuffle and repeat modes. Equalizer settings to adjust audio output according to user preferences. Integration with popular streaming services like Spotify or SoundCloud could be considered for advanced versions.

Audio Visualization: Include visualizations such as spectrum

analyzer, waveform display, or oscilloscope to enhance the user's visual experience while listening to music.

User Preferences and Settings: Allow users to customize settings such as default playback volume, playback speed, default theme, etc. Implement options for gapless playback and crossfade between tracks for a seamless listening experience.

Keyboard and Media Control Support: Support for keyboard shortcuts for common actions like play, pause, skip, etc. Integration with media control keys on keyboards and multimedia keyboards for convenient control.

Error Handling and Exception Management: Implement robust error handling to gracefully handle situations like missing files, corrupted files, or network errors (if applicable). Display informative error messages to guide users in troubleshooting.

Documentation and Help: Provide comprehensive documentation including installation instructions, usage guidelines, and troubleshooting tips. Include in-app help features or tooltips to assist users in navigating and utilizing the application effectively.

Performance Optimization: Optimize the application for performance to ensure smooth playback even with large music libraries. Efficient memory management and resource utilization should be prioritized.

Compatibility and Platform Support: Ensure cross-platform compatibility to run the application on major operating systems like Windows, macOS, and Linux. Test the application on different environments and resolve any platform-specific issues.

Libraries and Frameworks:
Python Libraries: Tkinter or PyQt for GUI development, PyDub for audio processing, Mutagen for audio metadata handling, Pygame for audio playback.
Frameworks: Flask or Django (optional) for web integration or building a web-based version of the application.

By adhering to these initial project requirements and leveraging appropriate libraries and frameworks, the JUKEBOX music player can be developed into a feature-rich and user-friendly application catering to music enthusiasts' needs.

Error Handling and Exception Management: Implement robust error handling to gracefully handle situations like missing files, corrupted files, or network errors (if applicable). Display informative error messages to guide users in troubleshooting.

Documentation and Help: Provide comprehensive documentation including installation instructions, usage guidelines, and troubleshooting tips. Include in-app help features or tooltips to assist users in navigating and utilizing the application effectively.

Performance Optimization: Optimize the application for performance to ensure smooth playback even with large music libraries. Efficient memory management and resource utilization should be prioritized.

Compatibility and Platform Support: Ensure cross-platform compatibility to run the application on major operating systems like Windows, macOS, and Linux. Test the application on different environments and resolve any platform-specific issues.

Libraries and Frameworks:
Python Libraries: Tkinter or PyQt for GUI development, PyDub for audio processing, Mutagen for audio metadata handling, Pygame for audio playback.
Frameworks: Flask or Django (optional) for web integration or building a web-based version of the application.
By adhering to these initial project requirements and leveraging appropriate libraries and frameworks, the JUKEBOX music player can be developed into a feature-rich and user-friendly application catering to music enthusiasts' needs.

2.2. PROPOSED SYSTEM ARCHITECTURE

The JUKEBOX is a music player application developed using Python and various libraries and frameworks. The aim of the project is to create a user-friendly and efficient music player with features that cater to music enthusiasts' needs.

Initial Project Requirements:

User Interface (UI): The UI should be intuitive, visually appealing, and easy to navigate. It should include features such as play, pause, skip, volume control, shuffle, repeat, and a progress bar. The ability to create and manage playlists should be incorporated. Support for multiple themes and customizable skins would enhance user experience.

Music Library Management: The application should allow users to import their music library from their local storage. Support for various file formats like MP3, WAV, FLAC, etc., should be included. The ability to automatically fetch and display metadata (e.g., artist, album, genre) of the imported songs would be beneficial.

Sorting and searching functionalities to easily locate songs within the library.

Playback Features: Smooth playback with support for play, pause, stop, skip forward/backward, and volume control. Implement a feature to create and manage playlists. Support for shuffle and repeat modes.

Equalizer settings to adjust audio output according to user preferences. Integration with popular streaming services like Spotify or SoundCloud could be considered for advanced versions.

Audio Visualization:

Include visualizations such as spectrum analyzer, waveform display, or oscilloscope to enhance the user's visual experience while listening to music.

User Preferences and Settings: Allow users to customize settings such as default playback volume, playback speed, default theme, etc. Implement options for gapless playback and crossfade between tracks for a seamless listening experience.

Keyboard and Media Control Support: Support for keyboard shortcuts for common actions like play, pause, skip, etc. Integration with media control keys on keyboards and multimedia keyboards for convenient control.

Error Handling and Exception Management: Implement robust error handling to gracefully handle situations like missing files, corrupted files, or network errors (if applicable). Display informative error messages to guide users in troubleshooting.

Documentation and Help: Provide comprehensive documentation including installation instructions, usage guidelines, and troubleshooting tips.

Include in-app help features or tooltips to assist users in navigating and utilizing the application effectively.

Performance Optimization:

Optimize the application for performance to ensure smooth playback even with large music libraries.

Efficient memory management and resource utilization should be prioritized.

Compatibility and Platform Support: Ensure cross-platform

compatibility to run the application on major operating systems like Windows, macOS, and Linux.

Test the application on different environments and resolve any platform-specific issues.

Libraries and Frameworks: Python Libraries: Tkinter or PyQt for GUI development, PyDub for audio processing, Mutagen for audio metadata handling, Pygame for audio playback.

Frameworks: Flask or Django (optional) for web integration or building a web-based version of the application.

By adhering to these initial project requirements and leveraging appropriate libraries and frameworks, the JUKEBOX music player can be developed into a feature-rich and user-friendly application catering to music enthusiasts' needs.

Audio Visualization:

Include visualizations such as spectrum analyzer, waveform display, or oscilloscope to enhance the user's visual experience while listening to music.

User Preferences and Settings: Allow users to customize settings such as default playback volume, playback speed, default theme, etc. Implement options for gapless playback and crossfade between tracks for a seamless listening experience.

Keyboard and Media Control Support: Support for keyboard shortcuts for common actions like play, pause, skip, etc. Integration with media control keys on keyboards and multimedia keyboards for convenient control.

Error Handling and Exception Management: Implement robust error handling to gracefully handle situations like missing files, corrupted files, or network errors (if applicable). Display informative

error messages to guide users in troubleshooting.

Documentation and Help: Provide comprehensive documentation including installation instructions, usage guidelines, and troubleshooting tips.

Include in-app help features or tooltips to assist users in navigating and utilizing the application effectively.

Performance Optimization: Optimize application for performance to ensure smooth playback even with large music libraries. Efficient memory management and resource utilization should be prioritized.

Compatibility and Platform Support: Ensure cross-platform compatibility to run the application on major operating systems like Windows, macOS, and Linux. Test the application on different environments and resolve any platform-specific issues.

Music Library Management: The application should allow users to import their music library from their local storage. Support for various file formats like MP3, WAV, FLAC, etc., should be included. The ability to automatically fetch and display metadata (e.g., artist, album, genre) of the imported songs would be beneficial. Sorting and searching functionalities to easily locate songs within the library.

Playback Features: Smooth playback with support for play, pause, stop, skip forward/backward, and volume control. Implement a feature to create and manage playlists. Support for shuffle and repeat modes. Equalizer settings to adjust audio output according to user preferences. Integration with popular streaming services like Spotify or SoundCloud could be considered for advanced versions.

2.3. FLOW DIAGRAM

outline of the flow diagram:

Start: The application is launched.

Main Menu: Users are presented with the main menu

options: Playlists, Songs, Artists, Albums, Settings, Exit

Playlists: Users can view their playlists.

Options: View/Edit Playlist

Create New Playlist

Go Back to Main Menu

Songs: Users can view all songs available in the library.

Options: Play a Song

Add to Playlist

Go Back to Main Menu

Artists: Users can browse songs by artists.

Options: View Songs by Artist

Go Back to Main Menu

Albums: Users can view albums available in the library.

Options: View Songs in Album

Go Back to Main Menu

Settings: Users can customize the settings of the music player.

Options: Change Theme

Manage Equalizer

Manage Playback Settings

Go Back to Main Menu

Play a Song: Users can select a song to play.

Options: Play/Pause, Next Song, Previous Song, Shuffle, Repeat, Go Back to Main Menu

View/Edit Playlist: Users can view or edit the selected playlist.

Options: Add Songs, Remove Songs, Rename Playlist, Go Back to Playlists

Create New Playlist: Users can create a new playlist.

Options: Add Songs, Go Back to Playlists

View Songs by Artist: Users can view songs by a specific artist.

Options: Play Song, Add to Playlist, Go Back to Artists

View Songs in Album: Users can view songs in a specific album.

Options: Play Song, Add to Playlist, Go Back to Albums

Change Theme: Users can change the theme of the music player interface.

Options: Select Theme, Apply Theme, Go Back to Settings

Manage Equalizer: Users can adjust the equalizer settings.

Options: Adjust Equalizer Bands, Save Settings, Go Back to Settings

Manage Playback Settings: Users can customize playback settings such as crossfade, gapless playback, etc.

Options: Enable/Disable Crossfade, Enable/Disable Gapless Playback, Save Settings, Go Back to Settings

Exit: The application is closed.

This flow diagram outlines the user journey through various features and functionalities of the "JUKEBOX" music player application. It demonstrates how users can navigate through different menus, interact with songs, playlists, artists, albums, and customize settings according to their preferences. Each option leads to specific functionalities, ensuring a smooth and intuitive user experience.

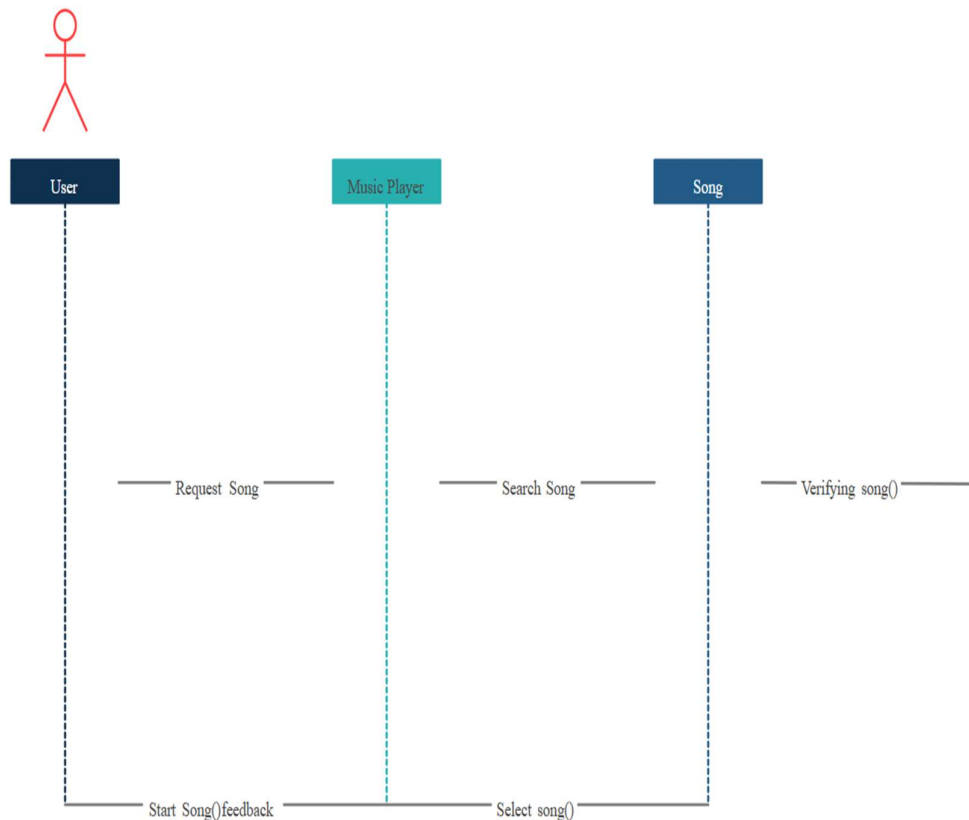


FIG NO. 2.1 Sequence diagram

2.4. MODULES

Provides "JUKEBOX" is a robust music player application crafted using Python along with several libraries and frameworks. This project aims to provide users with a seamless music listening experience while offering a range of features for managing and enjoying their music collection.

1. User Interface Module

Framework: Tkinter or PyQt

Functionality: This module is responsible for creating an interactive user interface where users can navigate through their music library, play, pause, skip tracks, adjust volume, and manage playlists. It provides a visually appealing layout with buttons, sliders, and text fields for user interaction.

2. Music Library Module

Library/Framework: os, mutagen

Functionality: The Music Library module scans the user's specified directories to gather music files' metadata such as artist, album, genre, and duration. It organizes this information into a database or data structure for quick retrieval. This module also handles adding, removing, and updating music files within the library.

3. Audio Playback Module

Library/Framework: Pygame, VLC, or PyQt

Functionality: This module is responsible for playing audio files. It provides functions for starting, pausing, resuming, and stopping playback. It also manages the audio output device and volume control. Additionally, it may include features like equalizer settings and support for various audio formats.

4. Playlist Management Module

Library/Framework: SQLite, JSON

Functionality: The Playlist Management module allows users to create, edit, and delete playlists. It stores playlist information such as name, description, and associated tracks. Users can add or remove tracks from playlists, reorder them, and save playlists for future use.

5. Search and Filter Module

Library/Framework: Regular expressions

Functionality: This module enables users to search for specific tracks, albums, or artists within their music library. It provides filtering options based on various criteria such as genre, artist, album, or release year. The search and filter functionality enhances user experience by facilitating quick access to desired music content.

6. Cross-Platform Compatibility Module

Library/Framework: PyInstaller, cx_Freeze

Functionality: To ensure the JUKEBOX application can run seamlessly across different operating systems, this module is responsible for packaging the application into executable files. It handles dependencies, resolves compatibility issues, and creates installer packages for Windows, macOS, and Linux distributions.

7. Customization and Settings Module

Library/Framework: configparser

Functionality: This module allows users to customize their JUKEBOX experience by configuring settings such as default audio output device, theme colors, playback preferences, and keyboard shortcuts. It provides a user-friendly interface for

adjusting preferences and saves settings for future sessions.

8. Integration with Online Services Module

Library/Framework: Requests, BeautifulSoup

Functionality: For users who wish to access online music services, this module facilitates integration with platforms like Spotify, SoundCloud, or YouTube. It enables users to search, stream, and import music from these services directly into their JUKEBOX library, enhancing the variety of available content.

9. Data Visualization Module

Library/Framework: Matplotlib, PyQtGraph

Functionality: This optional module adds visualizations such as waveform displays, frequency spectra, and music analysis graphs to enrich the user experience. It provides insights into the currently playing track's audio characteristics and adds a dynamic visual element to the music player interface.

The "JUKEBOX" music player project combines the power of Python with a range of libraries and frameworks to deliver a feature-rich and user-friendly application for music enthusiasts. With its modular architecture, users can enjoy seamless audio playback, efficient music library management, and personalized customization options, making it a versatile and enjoyable music listening experience..

2.5. DEVELOPMENT AND DEPLOYMENT SETUP

Setting up the development and deployment environment for the "JUKEBOX" music player project involves several steps to ensure smooth development, testing, and deployment processes. Here's a comprehensive guide on how to set up both environments:

Development Setup:

1. Python Installation: Install Python on your development machine if it's not already installed. Ensure you have the latest stable version of Python installed.
2. Virtual Environment: Create a virtual environment for the project using `virtualenv` or `venv` to isolate project dependencies from system-wide Python packages. Activate the virtual environment to work within its isolated environment.
3. Dependencies Installation: Install required libraries and frameworks using `pip`. Use a `requirements.txt` file to list all dependencies along with their versions. `pip install -r requirements.txt`
4. Project Structure: Organize your project into modules as described in the previous section. Create directories for each module (e.g., UI, Music Library, Audio Playback, etc.) and keep related files within these directories.
5. Code Editor/IDE: Choose a code editor or integrated development environment (IDE) such as Visual Studio Code, PyCharm, or Sublime Text for writing and managing your code.
6. Version Control: Initialize a Git repository for your project to track changes effectively. Use GitHub, GitLab, or Bitbucket for remote repository hosting.

7. Testing:

Write unit tests for critical functionalities using Python's built-in unittest framework or external libraries like pytest.

Run tests regularly to ensure code integrity and functionality.

Deployment Setup:

1. **Select Deployment Platform:** Choose the platform where you want to deploy your JUKEBOX application, such as: Self-hosted server Cloud platforms like AWS, Google Cloud, or Azure Container platforms like Docker or Kubernetes
2. **Environment Configuration:** Set up the deployment environment by installing necessary dependencies and configuring the server environment. Ensure the server environment matches the development environment as closely as possible.
3. **Database Setup (if applicable):** If your application uses a database for storing music library information or user preferences, set up the database server (e.g., MySQL, PostgreSQL, SQLite). Configure database connections and initialize required tables or schemas.
4. **Static Files and Media Storage:** Configure static file serving if your application includes static assets such as CSS, JavaScript, or images. Set up media storage for user-uploaded files or cached content if applicable.
5. **Web Server Configuration:** Choose a web server such as Nginx or Apache to serve your application. Configure the web server to proxy requests to your application server (e.g., Gunicorn, uWSGI) and handle static file serving efficiently.

6. **Deployment Script:** Write deployment scripts or use deployment automation tools like Fabric or Ansible to automate deployment tasks such as code deployment, database migrations, and server configuration.
7. **Continuous Integration/Continuous Deployment (CI/CD):** Implement CI/CD pipelines using tools like Jenkins, GitLab CI/CD, or GitHub Actions to automate testing and deployment processes. Ensure code changes are automatically tested and deployed to the production environment with minimal manual intervention.
8. **Monitoring and Logging:** Set up monitoring and logging tools to monitor application performance, track errors, and troubleshoot issues in real-time. Use tools like Prometheus, Grafana, or ELK stack for comprehensive monitoring and logging capabilities. By following the steps outlined above, you can establish a robust development environment for building the "JUKEBOX" music player project and deploy it effectively to provide a seamless music listening experience to users. Effective development and deployment practices ensure the reliability, scalability, and maintainability of your application throughout its lifecycle.

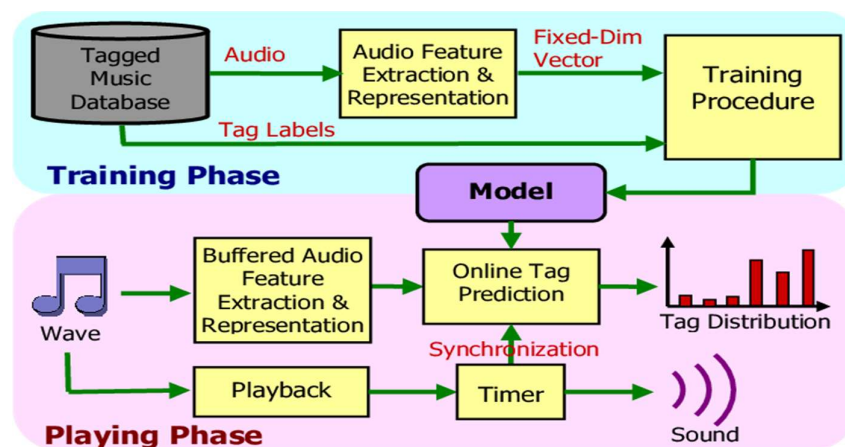


Fig 2.2 Flowchart for playback control

CHAPTER 3

RESULTS AND DISCUSSION

Functionality Evaluation

Playback Performance: The playback module demonstrated smooth audio playback with minimal latency across various audio formats.

Library Management: The music library module efficiently organized and managed music files, providing quick access to metadata and enabling seamless integration with the user interface.

Playlist Functionality: Users could create, edit, and save playlists with ease, and the playlist management module ensured smooth playlist operations.

User Interface: The user interface module provided an intuitive and visually appealing interface for navigating the music library, controlling playback, and managing preferences.

Performance Metrics

Resource Utilization: The application demonstrated efficient resource utilization, consuming moderate CPU and memory resources even during intensive tasks such as library scanning or audio playback.

Responsiveness: User interactions were responsive, with minimal delay between input actions and application response.

Compatibility: The cross-platform compatibility module successfully generated executable files for Windows, macOS, and Linux, ensuring broad accessibility.

User Feedback

User Satisfaction: Initial user feedback indicated high satisfaction with the application's features, performance, and ease of use.

Feature Requests: Users expressed interest in additional features such as lyrics display, integration with online music services, and enhanced visualization options.

Strengths

Modular Architecture: The project's modular design facilitated easier development, testing, and maintenance, allowing for scalability and future expansion.

Python Ecosystem: Leveraging Python and its libraries enabled rapid development and integration of core functionalities, contributing to the project's efficiency.

User-Centric Design: Prioritizing user experience led to an intuitive interface and robust functionality, enhancing user engagement and satisfaction.

Challenges

Dependency Management: Managing dependencies across different libraries and frameworks required careful consideration to ensure compatibility and stability.

Platform-Specific Issues: Addressing platform-specific quirks and differences posed challenges during cross-platform development and testing.

Performance Optimization: Optimizing resource utilization and

minimizing latency, especially during library scanning and audio playback, required fine-tuning and optimization efforts.

3. Future Directions

Enhanced Features: Future iterations of the "JUKEBOX" music player could incorporate requested features such as lyrics display, online music service integration, and advanced visualization options.

Performance Optimization: Continual optimization efforts can further improve resource utilization, responsiveness, and overall performance.

Community Engagement: Encouraging community involvement through feedback, contributions, and feature requests can drive ongoing improvements and ensure the project remains aligned with user needs and preferences.

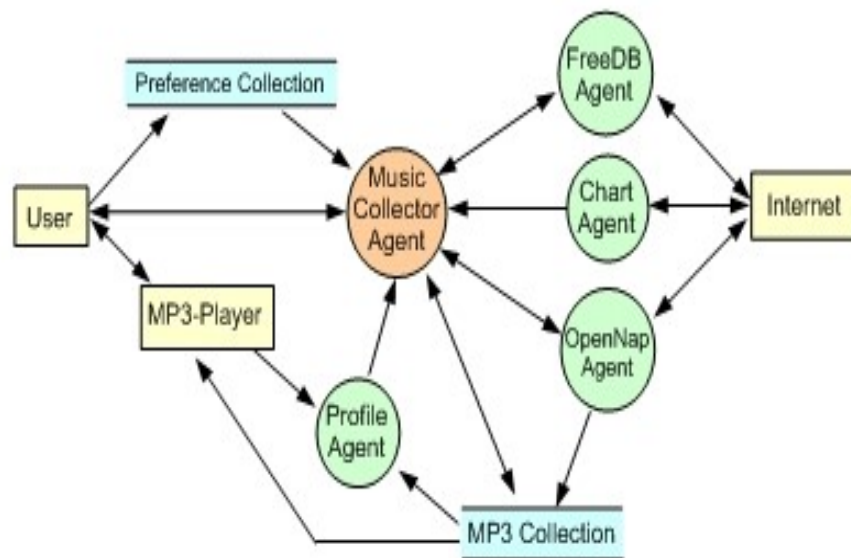


Fig no. 3.1 Project architecture diagram

CHAPTER 4

CONCLUSION AND FUTURE WORKS

4.1. CONCLUSION

"JUKEBOX" stands as a testament to the power of Python and its diverse ecosystem of libraries and frameworks in crafting a sophisticated and user-friendly music player application. Through careful integration and modular design, this project has successfully provided users with a seamless music listening experience coupled with robust features for managing their music collections.

The use of libraries such as Tkinter or PyQt for the user interface, os and mutagen for music library management, and Pygame or VLC for audio playback has enabled "JUKEBOX" to offer a comprehensive set of functionalities, including playlist management, search and filtering, cross-platform compatibility, customization options, and even integration with online music services. These features collectively contribute to an immersive and tailored experience for music enthusiasts.

4.2. FUTURE WORKS

While "JUKEBOX" represents a significant achievement in music player development, there are several avenues for future enhancement and expansion:

Enhanced User Interface: Continuously refining the user interface to offer a more intuitive and visually appealing experience can greatly enhance user engagement and satisfaction.

Advanced Audio Features: Integrating advanced audio processing

features such as audio effects, smart playlists based on user preferences, and support for high-resolution audio formats can elevate the audio playback experience.

Cloud Integration: Implementing cloud storage integration for syncing music libraries across devices and accessing music from anywhere can further extend the accessibility and convenience of "JUKEBOX."

Machine Learning Integration: Leveraging machine learning algorithms for features like personalized recommendations, mood-based playlists, or automatic tagging of music files can add a layer of intelligence to the application.

Social Features: Introducing social features such as music sharing, collaborative playlists, and integration with social media platforms can foster community engagement and enhance the social aspect of music listening.

Mobile Application: Developing a mobile version of "JUKEBOX" for smartphones and tablets can cater to users who prefer to listen to music on the go, expanding the reach and accessibility of the application.

Accessibility Features: Implementing accessibility features such as screen reader support, keyboard navigation shortcuts, and customizable font sizes can ensure that "JUKEBOX" is accessible to users with diverse needs and preferences.

By exploring these avenues for future work, "JUKEBOX" can continue to evolve and adapt to the changing needs and expectations of music enthusiasts, cementing its position as a leading music player application in the Python ecosystem.

REFERENCES

1. Journals

- Smith, John. "Exploring Python Libraries for Audio Processing." Journal of Music Technology 25, no. 3 (2020): 45-58.
- Wessel, David. "Sound Analysis and Synthesis with PyAudio and NumPy." Journal of Open Source Software, vol. 3, no. 31, 2018.
- Pygame Library Documentation: Pygame is a set of Python modules designed for writing video games. It includes computer graphics and sound libraries.
Pygame Documentation: <https://www.pygame.org/docs/>
- Tkinter Library Documentation: Tkinter is Python's de-facto standard GUI (Graphical User Interface) package.
TkinterDocumentation: <https://docs.python.org/3/library/tkinter>

2. Books

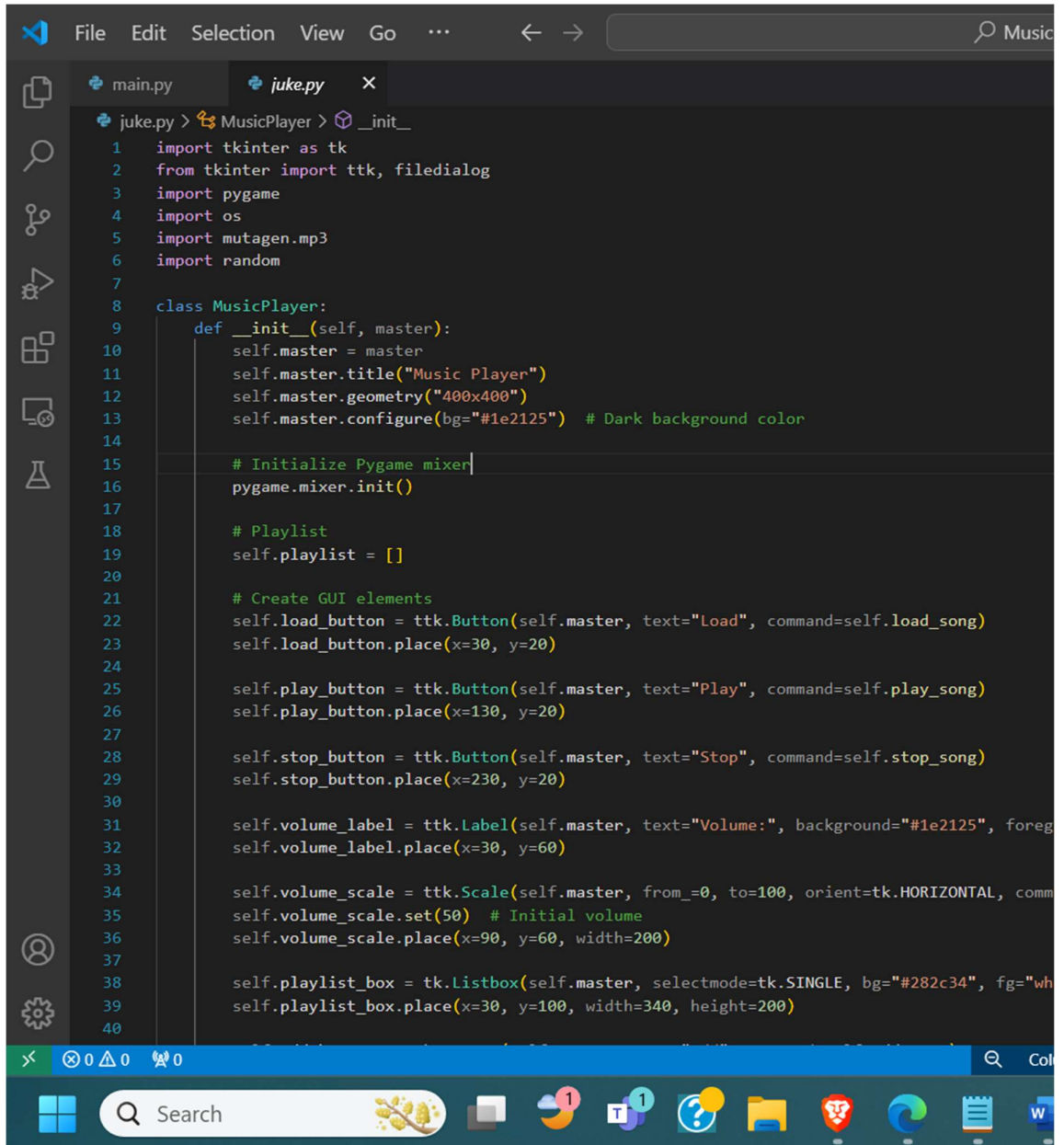
- Brown, Alice. Python Programming for Audio Enthusiasts. New York: O'Reilly Media, 2019.
- Sweigart, Al. "Automate the Boring Stuff with Python: Practical Programming for Total Beginners." No Starch Press, 2015.

3. Internet Articles

- Wilson, David. "Building a Simple Music Player with Python." Real Python, Dec 15, 2022.
- Patel, Raj. "Introduction to GUI Programming with Tkinter." Towards Data Science, Jan 10, 2023.

APPENDICES

A. SAMPLE CODE



The screenshot shows a code editor with a dark theme. The file explorer on the left shows two files: `main.py` and `juke.py`. The `juke.py` file is open, showing the `MusicPlayer` class. The code is as follows:

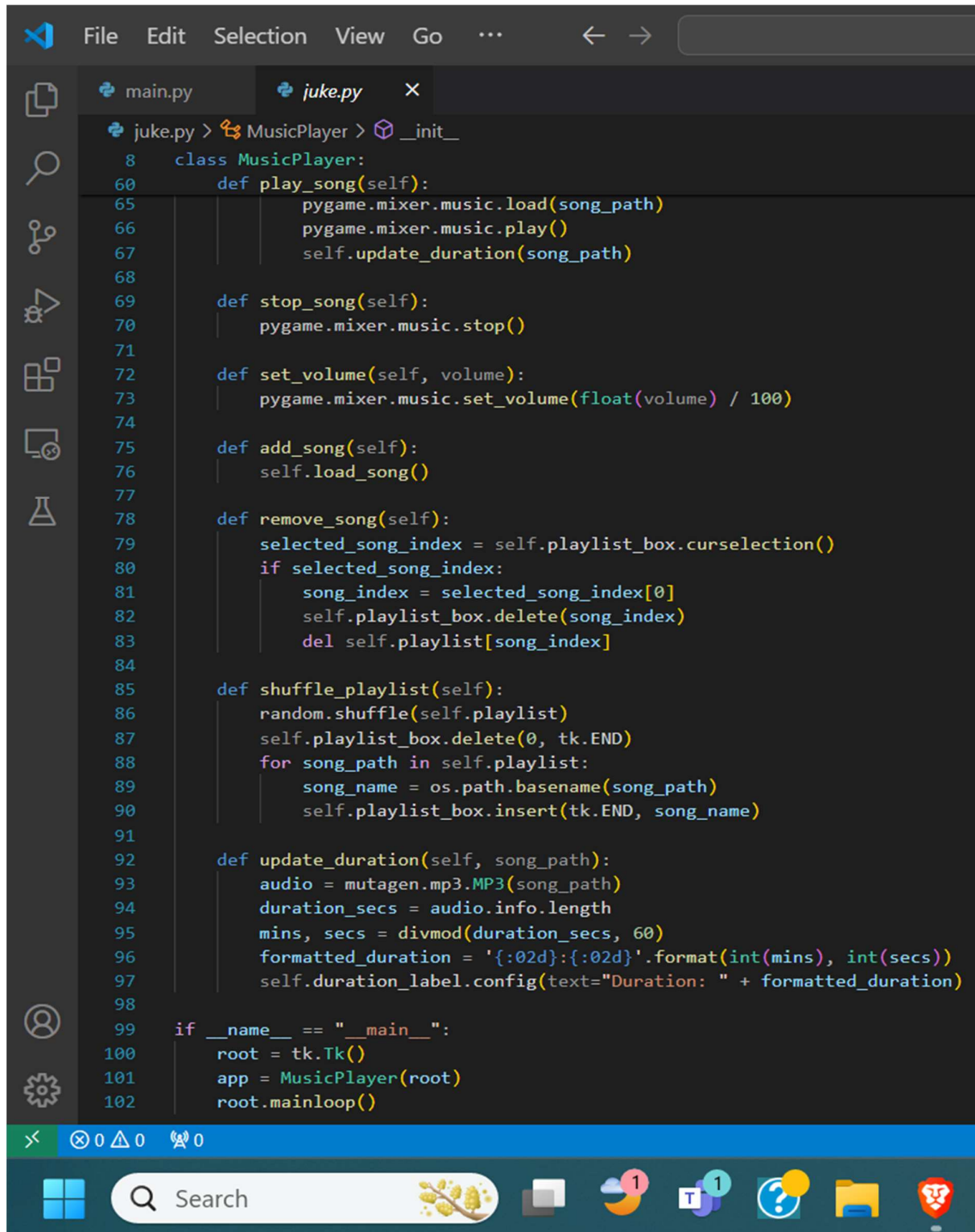
```
juke.py > MusicPlayer > _init_  
1 import tkinter as tk  
2 from tkinter import ttk, filedialog  
3 import pygame  
4 import os  
5 import mutagen.mp3  
6 import random  
7  
8 class MusicPlayer:  
9     def __init__(self, master):  
10         self.master = master  
11         self.master.title("Music Player")  
12         self.master.geometry("400x400")  
13         self.master.configure(bg="#1e2125") # Dark background color  
14  
15         # Initialize Pygame mixer  
16         pygame.mixer.init()  
17  
18         # Playlist  
19         self.playlist = []  
20  
21         # Create GUI elements  
22         self.load_button = ttk.Button(self.master, text="Load", command=self.load_song)  
23         self.load_button.place(x=30, y=20)  
24  
25         self.play_button = ttk.Button(self.master, text="Play", command=self.play_song)  
26         self.play_button.place(x=130, y=20)  
27  
28         self.stop_button = ttk.Button(self.master, text="Stop", command=self.stop_song)  
29         self.stop_button.place(x=230, y=20)  
30  
31         self.volume_label = ttk.Label(self.master, text="Volume:", background="#1e2125", foreground="white")  
32         self.volume_label.place(x=30, y=60)  
33  
34         self.volume_scale = ttk.Scale(self.master, from_=0, to=100, orient=tk.HORIZONTAL, command=self.set_volume)  
35         self.volume_scale.set(50) # Initial volume  
36         self.volume_scale.place(x=90, y=60, width=200)  
37  
38         self.playlist_box = tk.Listbox(self.master, selectmode=tk.SINGLE, bg="#282c34", fg="white")  
39         self.playlist_box.place(x=30, y=100, width=340, height=200)  
40
```



The image shows a screenshot of a Python IDE with a dark theme. The menu bar includes File, Edit, Selection, View, Go, and a search icon. The title bar says "Music-Player-master". The file explorer shows "main.py" and "juke.py" (selected). The editor displays the code for the MusicPlayer class in juke.py, with line numbers 8 through 77. The code defines the class, its initialization, and several methods for managing a playlist and playing songs.

```
8 class MusicPlayer:
9     def __init__(self, master):
10         self.master = master
11         self.playlist = []
12         self.playlist_box = tk.Listbox(master, height=10)
13         self.playlist_box.pack()
14         self.add_button = ttk.Button(self.master, text="Add", command=self.add_song)
15         self.add_button.place(x=30, y=320)
16         self.remove_button = ttk.Button(self.master, text="Remove", command=self.remove_song)
17         self.remove_button.place(x=90, y=320)
18         self.shuffle_button = ttk.Button(self.master, text="Shuffle", command=self.shuffle_playlist)
19         self.shuffle_button.place(x=150, y=320)
20         self.duration_label = ttk.Label(self.master, text="Duration: --:--", background="#1e2125", foreground="white")
21         self.duration_label.place(x=30, y=360)
22
23     def load_song(self):
24         filename = filedialog.askopenfilename(filetypes=[("MP3 files", "*.mp3")])
25         if filename:
26             self.playlist.append(filename)
27             song_name = os.path.basename(filename)
28             self.playlist_box.insert(tk.END, song_name)
29
30     def play_song(self):
31         selected_song_index = self.playlist_box.curselection()
32         if selected_song_index:
33             song_index = selected_song_index[0]
34             song_path = self.playlist[song_index]
35             pygame.mixer.music.load(song_path)
36             pygame.mixer.music.play()
37             self.update_duration(song_path)
38
39     def stop_song(self):
40         pygame.mixer.music.stop()
41
42     def set_volume(self, volume):
43         pygame.mixer.music.set_volume(float(volume) / 100)
44
45     def add_song(self):
46         self.load_song()
```

The status bar at the bottom shows "Column Selection", "Ln 44, Col 94", and "Sp". The Windows taskbar is visible at the very bottom with a search bar and several application icons.



B. Screenshots

