PROJECT REPORT

“Modern Desktop Audio Player”

(A Python-based Music Management and Playback System)

Submitted By:

Vedika Rajpali Reg. No.: `25BCE11386`

Branch: B.Tech – Computer Science & Engineering

Course Code: CSE1021

Course Name: Problem Solving and Programming

Submitted To:

[Faculty Name] School of Computing Science & Engineering

Academic Year: 2024 – 2025

1. Introduction

Music and audio playback applications are fundamental tools for digital consumption. The need for a lightweight, feature-rich, and locally focused audio player remains essential for users managing personal music libraries. Existing commercial solutions often rely heavily on streaming and network connectivity, overlooking the need for simple, offline playback and library management.

The **Modern Desktop Audio Player** is a dedicated application built using Python, the Tkinter framework for the Graphical User Interface (GUI), and the `pygame.mixer` module for robust audio handling. This project aims to provide users with essential features like playlist management, audio controls (play, pause, stop, volume), and track metadata display.

This project improves the user experience for managing local media files, offering an efficient and reliable tool free from complex online services.

2. Problem Statement

Users with large, locally stored music libraries often face challenges with existing software that is either too resource-intensive, overly complex, or primarily designed for streaming services. A simple desktop application is needed that can efficiently load, manage, and play various local audio formats (`.mp3` , `.wav` , `.ogg`) while providing an intuitive, responsive user interface.

The Modern Desktop Audio Player solves this by offering a dedicated, non-streaming, locally-driven solution for high-quality audio playback and basic library organization, addressing the gap between

commercial streaming platforms and basic operating system media players.

3. Project Objectives

- To design a minimal and aesthetically pleasing Graphical User Interface (GUI) for an audio player using Python and Tkinter.
- To implement core audio playback functions (Play, Pause, Stop, Seek, Volume Control) using the `pygame.mixer` module.
- To enable basic library management, including loading individual files and creating/saving simple music playlists.
- To display real-time track metadata (Title, Artist, Duration) extracted from audio files.
- To ensure the application is portable and runs reliably on standard desktop operating systems.

4. Functional Requirements

4.1 Audio Playback

- Play, Pause, Resume, and Stop functionality.
- Variable volume control (Slider).
- Seeking functionality within the current track.
- Support for common audio formats (MP3, WAV, OGG).

4.2 Library Management

- Button/Menu to load individual audio files.
- Ability to load and save simple playlists (e.g., as `.txt` or `.json` file paths).
- Display of the current playlist queue.
- Next/Previous track navigation.

4.3 User Interface & Metadata

- Display of current track title and duration.
- Status bar indication (Playing, Paused, Stopped).
- Responsive UI design using the Tkinter grid or pack manager.

5. Non-Functional Requirements

5.1 Usability

- Clean, intuitive layout with clearly labeled controls.
- Visual feedback on button presses and player status.

5.2 Performance

- Audio playback must be gapless and smooth, without stuttering.
- Library loading operations should be quick for typical directories.

5.3 Maintainability

- Code divided into logical classes (e.g., `Player`, `GUI`, `PlaylistManager`).

5.4 Reliability

- Robust error handling for corrupted or unsupported audio files.
- Player state must be preserved during window resizing.

6. System Architecture

The application follows a **Model-View-Controller (MVC)** pattern to separate the core logic from the user interface. This is implemented using Python modules.

6.1 Modular Architecture

- **Player Model (`player.py`):** The core engine. Handles `pygame.mixer` initialization, audio loading, and direct control over playback state (play/pause/stop/volume).
- **GUI View (`app.py`):** The user interface layer (Tkinter). Creates all visual elements (buttons, sliders, display) and captures user input events.
- **Controller (`app.py`):** Mediates between the View and the Model. It receives events from the GUI and calls appropriate methods in the Player Model.
- **Playlist Manager (`playlist.py`):** Handles file path storage, reading metadata, and sequential track selection.

7. Design Diagrams

7.1 Use Case Diagram

The user (Listener) interacts with the system to manage and play audio.

7.2 Class/Component Diagram

The Class Diagram shows the key classes and their relationship.

7.3 Data Structure (Playlist Table)

The core data structure is managed by the `PlaylistManager` and conceptually stores track information:

Field Name	Data Type / Purpose

`track_path`	`STRING` : Full file system path to the audio file
`track_title`	`STRING` : Extracted metadata (ID3 Title)
`track_artist`	`STRING` : Extracted metadata (ID3 Artist)
`duration`	`FLOAT` : Track duration in seconds

7.4 Sequence Diagram

Illustrates the flow of control when the user presses the **Play** button:

7.5 Workflow Diagram

The primary workflow for loading and playing a new track is as follows:

1. **Start:** Application Launch
2. **Action:** User Selects 'Load File'
3. **GUI:** Opens File Dialog, returns file path
4. **Controller:** Validates file type
5. **Playlist Manager:** Adds track to queue
6. **Metadata Extractor:** Reads ID3 tags
7. **Player Model:** Loads audio data using `pygame.mixer`
8. **Controller:** Calls `Player.play()`
9. **GUI:** Updates Track Title and Status Display
10. **End:** Track is Playing

8. Design Decisions & Rationale

Component	Rationale
Python	Rapid development and rich ecosystem for multimedia (Pygame) and GUI (Tkinter)
Tkinter	Standard Python library, minimizing deployment dependencies and offering simplicity for a dedicated desktop application.
`pygame.mixer`	Optimized for fast, non-blocking, low-overhead audio playback, ensuring smooth performance.
MVC Pattern	Ensures separation of concerns (GUI vs. Audio Logic), improving maintainability and testability.

9. Implementation Details

9.1 Modules Overview

File	Purpose	Key Technology
`app.py`	Main GUI setup and event handling (Controller/View)	Tkinter
`player.py`	Core audio playback logic (Model)	`pygame.mixer`
`playlist.py`	Playlist creation, saving, and navigation	Python Lists / JSON
`metadata.py`	Extraction of ID3 tags (Title, Artist, Album)	`mutagen` (for ID3)

10. Testing Approach

10.1 Manual Testing

- **Playback Controls:** Testing Play/Pause/Stop/Skip functionality with various files.
- **Volume Control:** Testing the slider for continuous and boundary (0% and 100%) volume levels.
- **File Handling:** Loading various audio formats (MP3, WAV) and ensuring correct loading and playback.
- **Playlist Management:** Loading multiple files and ensuring track sequencing works correctly.

10.2 Edge Cases

- **Empty Playlist:** Starting the player with no files loaded.
- **Corrupted File:** Attempting to load an audio file that is invalid or corrupted.
- **Non-Audio File:** Attempting to load a text file or image (should raise an appropriate error).
- **Seeking during Pause:** Ensuring the seek bar updates correctly when paused and resumed.

10.3 Integration Testing

- **GUI-Model Link:** Verifying that button clicks correctly trigger the intended `pygame.mixer` function calls.
- **Playlist Update:** Ensuring that the UI list box updates immediately when a new file is loaded or the playlist is modified.

11. Challenges Faced

- **Thread Management:** Integrating the synchronous nature of Tkinter GUI events with the background audio process of `pygame.mixer` without freezing the UI. This was solved using dedicated threading for the volume update and seek bar.

- **ID3 Tagging:** Handling various audio file encodings and formats to reliably extract metadata (Artist, Title).
- **Cross-Platform Volume:** Ensuring the software volume control works uniformly across different operating systems.

12. Learnings & Key Takeaways

- **GUI Programming:** Gained experience in event-driven programming and managing UI components with Tkinter.
- **Multimedia Handling:** Practical application of a dedicated audio library (`pygame.mixer`).
- **Separation of Concerns (MVC):** Understanding the benefits of separating data, logic, and presentation layers for robust software design.
- **Error Resilience:** Implementing graceful failure mechanisms for file loading errors.

13. Future Enhancements

- **Database Integration:** Use SQLite (instead of JSON) for larger, more persistent music library indexing.
- **Visualizer:** Integrate a simple spectrum analyzer or audio visualization using the audio output stream.
- **Hotkeys:** Add keyboard shortcuts for Play/Pause/Skip functions.
- **Theming:** Implement a simple dark/light mode switcher for the UI.

14. References

- Tkinter Documentation