A PBL REPORT

ON

Design and Development of Music Application

Submitted to

INFORMATION TECHNOLOGY ,
BHARATI VIDYAPEETH (DEEMED TO BE UNIVERSITY) COLLEGE OF ENGINEERING, PUNE, INDIA

In Partial Fulfilment of the Requirement for the Award of

BACHELOR'S DEGREE IN INFORMATION TECHNOLOGY

BY

2214110233	HARSH AGARWAL
2214110238	ARYAN KUMAR
2214110270	RIYA SINGH
2214110288	BHUMI

UNDER THE GUIDANCE OF
Prof. Sampat Medhane

DEPARTMENT OF INFORMATION TECHNOLOGY

BHARATI VIDYAPEETH (DEEMED TO BE UNIVERSITY)

COLLEGE OF ENGINEERING, PUNE, INDIA - 411043

2024-2025

CERTIFICATE

This is to certify that the project entitled

Design and Development of Music Application

submitted by

2214110233 Harsh aggarwal

2214110238 Aryan Kumar

2214110270 Riya Singh

2214110288 Bhumi

is a record of bona fide work carried out by them, partially fulfilling the requirement for the Degree of Bachelor of Technology in Computer Engineering award at Bharati Vidyapeeth (Deemed to Be University) College of Engineering, Pune, India. This work is done during the academic year 2024-2025.

Signature

(Dr.Sandeep Vanjale)
(HOD, IT)

Signature

(Dr. Sampat Medhane)
(PBL Guide)

ACKNOWLEDGEMENT

We would like to express deepest appreciation towards Dr Vidula Sohoni, Principal, Bharati Vidyapeeth (Deemed to Be University) College of Engineering, Pune, India and Dr Sandeep Vanjale Head of Department of IT, who invaluable supported us in completing this project.

We are profoundly grateful to Dr. Sampat Medhane, Project guide for her expert guidance and continuous encouragement throughout to see that this project rights, its target since its commencement to its completion.

At last we must express our sincere heartfelt gratitude to all the staff members of IT Department who helped me directly or indirectly during this course of work.

Harsh Agarwal
Aryan Kumar
Riya Singh
Bhumi

ABSTRACT

In the modern era of digital music consumption, the development of a user-friendly and feature-rich music player application is essential for providing an enjoyable music experience. This abstract outline the design and development of a Windows application for a music player using the Python programming language.

The primary goal of this project is to create a versatile and user-friendly music player that allows users to organize, manage, and play their music collections seamlessly on the Windows operating system. The application is designed to support various audio formats, offer a user-friendly graphical user interface (GUI), and provide essential features such as playlist management, volume control, playback control, and song information display.

Key components of the project include:

- Graphical User Interface (GUI): The GUI is built using Python libraries such as Tkinter or PyQt, providing an intuitive interface for users to interact with the music player. It includes elements for browsing and selecting songs, displaying album artwork, and controlling playback.

- Audio Playback: The core of the music player handles audio playback. It supports multiple audio formats and includes features such as play, pause, stop, skip forward, skip backward, and volume control. The audio playback functionality is robust and ensures a smooth listening experience.

- Song Information: Detailed information about the currently playing song, including title, artist, album, and duration, is displayed to keep users informed about their music selection.

- Integration with Windows OS: The application is designed to seamlessly integrate with the Windows operating system, allowing users to access their music files and libraries effortlessly.

- Customization and User Preferences: Users can customize their music player experience by configuring settings such as playback options, visualizer settings, and visual themes.

The development process involves leveraging Python's versatility and rich ecosystem of libraries to create a stable and efficient music player application. The project prioritizes user experience, performance, and extensibility, ensuring that it can be further enhanced with additional features in the future.

Therefore, the development of a Windows music player application using Python demonstrates the power and flexibility of Python in building user-friendly desktop applications. The resulting music player offers an enjoyable and customizable music-listening experience while showcasing the capabilities of Python in software development.

TABLE OF CONTENTS

CHAPTER 1 - INTRODUCTION

Music, often described as the universal language, has played a pivotal role in human culture for centuries. In the digital age, music applications have become indispensable tools for both creators and consumers, revolutionizing how we interact with and enjoy music. Music applications have various advantages and a transformative impact on our musical experiences.

The music player allows you to play, pause, next, previous, and stop music at your convenience. We have provided the list of MP3 music files for selection and play according to you. We have created a visualizer which will sink according to the frequency of music. We have also provided a volume button to control volume. The code of our music player is also provided if you want to make any changes as per your choice.

Music applications have made it incredibly easy for anyone with a smartphone or computer to access a vast library of songs, albums, and playlists from virtually anywhere in the world. These applications offer an extensive and diverse selection of music genres, catering to all tastes and preferences. Many music applications offer the ability to download songs for offline listening. This is especially useful for users who want to enjoy music during commutes or in areas with limited internet connectivity. Several music apps offer high-definition audio streaming options, allowing audiophiles to enjoy their favorite tracks with exceptional sound quality, rivaling that of physical media. Many music applications seamlessly integrate with smart speakers, car systems, and other devices, enhancing accessibility and convenience across various platforms.

Thus, music applications have fundamentally transformed the way we consume and engage with music. They offer unparalleled accessibility, personalization, and convenience, making it easier to discover, enjoy, and share the beauty of music. As we delve deeper into our music applications, we will explore specific platforms, their unique features, and how they have contributed to the evolution of the music industry and our personal musical journeys.

Features of our music application :

1. Importing the required library modules.

2. Defining the functions for the application

3. Adding play, pause, next, stop, start and power buttons to operate.
4. MP3 music list.
5. Definition of function.
6. Closing the application
7. Retrieve music files from the file folder.
8. Creating the main window dialog box for the application
9. Adding the to the application
10. Adding the necessary widgets to the application and applying event triggers
11. Adding png files for button purposes.
12. Labels
13. Calling the functions
14. Use of GUI.
15. Added a visualizer.
16. Given the volume control button.

CHAPTER 2 – REQUIREMENT GATHERING

In response to the evolving landscape of music applications, we present this comprehensive **Software Requirements Specification (SRS)** document for the development of a cutting-edge **Music Application**. This document serves as the foundational blueprint that outlines the precise requirements, functionalities, and specifications necessary.

SRS is a formal report, that acts as a representation of software that enables the customers to review whether it (SRS) is according to their requirements. Also, it comprises user requirements for a system as well as detailed specifications of the system requirements.

Software Requirements Specification (SRS) for a Windows Music Player Application

1. INTRODUCTION

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to provide a comprehensive overview of the design and development of a Windows application for a music player using Python. It outlines the functional and non-functional requirements, interfaces, and constraints of the software.

1.2 Document Conventions

Bold Text: Represents section headings.

Italic Text: Represents placeholders or variable names.

Monospace Text: Represents code snippets, file names, or software components.

1.3 Intended Audience and Reading Suggestions

This document is intended for the development team, project managers, quality assurance team, and any stakeholders involved in the design and development of the music player application. Readers are encouraged to review the document in its entirety to understand the scope and requirements of the project.

1.4 Project Scope

The project aims to design and develop a user-friendly Windows music player application using the Python programming language. The application will support audio playback, playlist management, customization options, and integration with the Windows operating system. It is intended for individual users who want to manage and enjoy their music collections on Windows-based computers.

1.5 References

No external references are cited in this document.

2. OVERALL DESCRIPTION

2.1 Product Perspective

The music player application is a standalone software product that does not rely on external systems or dependencies. It is designed to run on Windows operating systems and will provide a user interface for managing and playing music files.

2.2 Product Features

The key features of the music player application include:

- Importing the required library modules.
- Defining the functions of the application
- Adding play, pause, next, stop, start, and power buttons to operate.
- MP3 music list.

- Definition of function.
- Closing the application.
- Retrieve music files from the file folder.
- Creating the main window dialog box for the application
- Adding to the application
- Adding the necessary widgets to the application and applying event triggers
- Adding png files for button purposes.
- Labels
- Calling the functions
- Use of GUI.
- Added a visualizer.
- Given the volume control button.

2.3 User Classes and Characteristics

The primary user class for this application includes:

End Users: Individuals who want to play and manage their music collection on a Windows computer. Users should have basic computer literacy.

2.4 Operating Environment

The application will operate in the following environment:

- Operating System: Windows 7, 8, 10, and later.
- Hardware: A standard Windows-based computer with sufficient processing power and memory for audio playback.

2.5 Design and Implementation Constraints

The application will be developed using Python and relevant libraries for GUI and audio playback.

It must adhere to Windows GUI guidelines for a consistent user experience.

Audio codec compatibility may be limited based on the availability of Python libraries.

2.6 User Documentation

User documentation, including user guides and installation instructions, will be provided along with the application.

2.7 Assumptions and Dependencies

- It is assumed that users have a valid Windows operating system.
- Dependencies include Python libraries for audio playback and GUI development.

3. SYSTEM FEATURES

3.1 System Feature 1

Feature: Audio Playback

Description: The application should allow users to play audio files in various formats, providing controls for play, pause, stop, volume adjustment, and seek functionality.

3.2 System Feature 2

Feature: Visualizer

Description: The application should allow users to visualize the audio through an integrated visualizer using the scales of music, and other graphical parameters.

4. EXTERNAL INTERFACE REQUIREMENTS

4.1 User Interfaces

The application will have a graphical user interface (GUI) designed for Windows, featuring menus, buttons, playlist views, and playback controls.

4.2 Hardware Interfaces

The application will utilize standard audio output and input interfaces available on Windows computers.

4.3 Software Interfaces

The application will interact with the Windows operating system to access music files.

Python libraries for audio playback and GUI development will be used.

4.4 Communications Interfaces

No external communication interfaces are required.

5. OTHER NON-FUNCTIONAL REQUIREMENTS

5.1 Performance Requirements

The application should provide responsive playback and interface interactions, even with a large music library.

Loading and switching between tracks should be efficient.

5.2 Safety Requirements

The application will not perform any actions that could harm the user's data or system.

5.3 Security Requirements

The application will not access or modify sensitive user data without explicit permission.

5.4 Software Quality Attributes

- The application should be user-friendly, with an intuitive and visually appealing interface.
- It should be robust, handling errors gracefully and providing clear error messages.
- It should be maintainable and extensible for future updates and enhancements.

6. OTHER REQUIREMENTS

No additional requirements are specified in this document.

Appendix A: Glossary

No glossary terms are included in this document.

Appendix B: Analysis Models

No analysis models are included in this document. **FIG. – 2.1**

- EER Diagram –

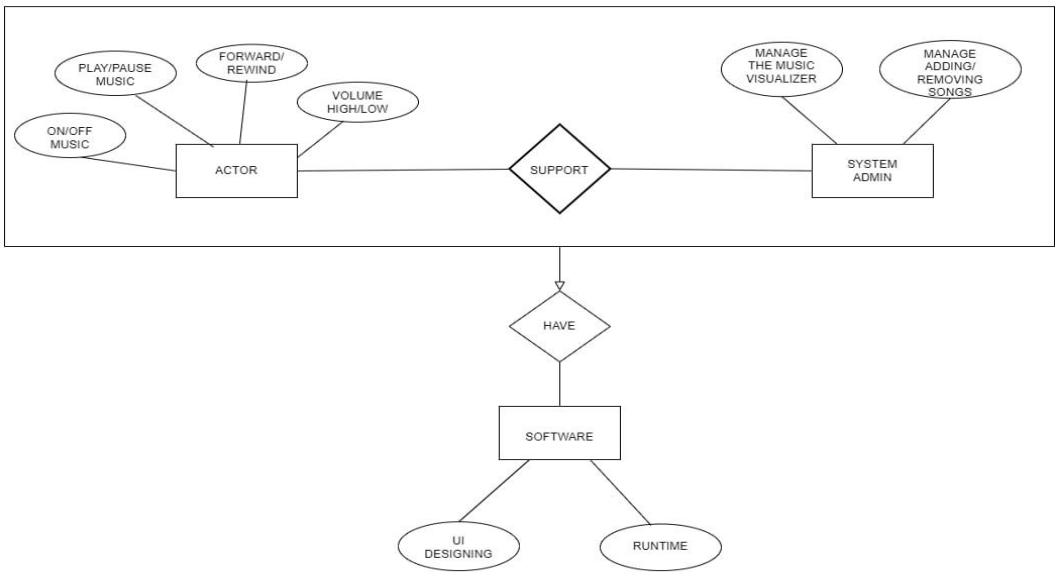
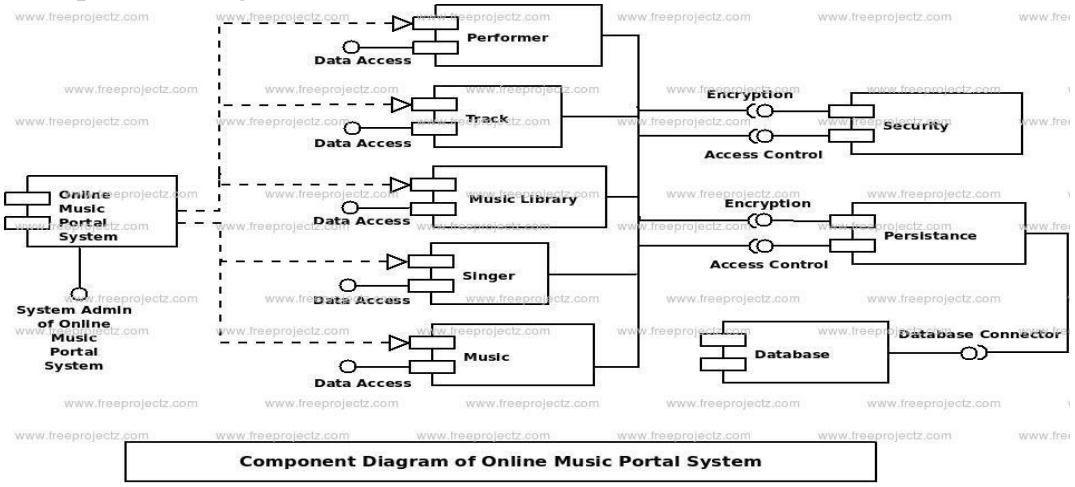


FIG. – 2.2

- Component diagram –



- Data flow diagram –

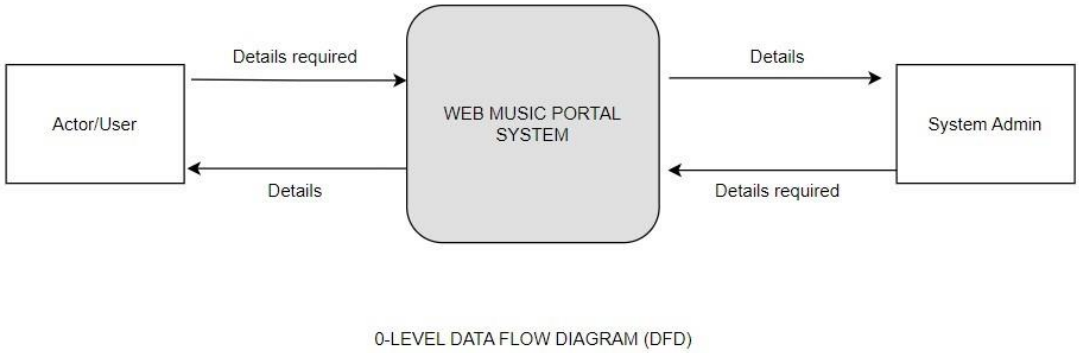


FIG. – 2.3

Appendix C: Issues List

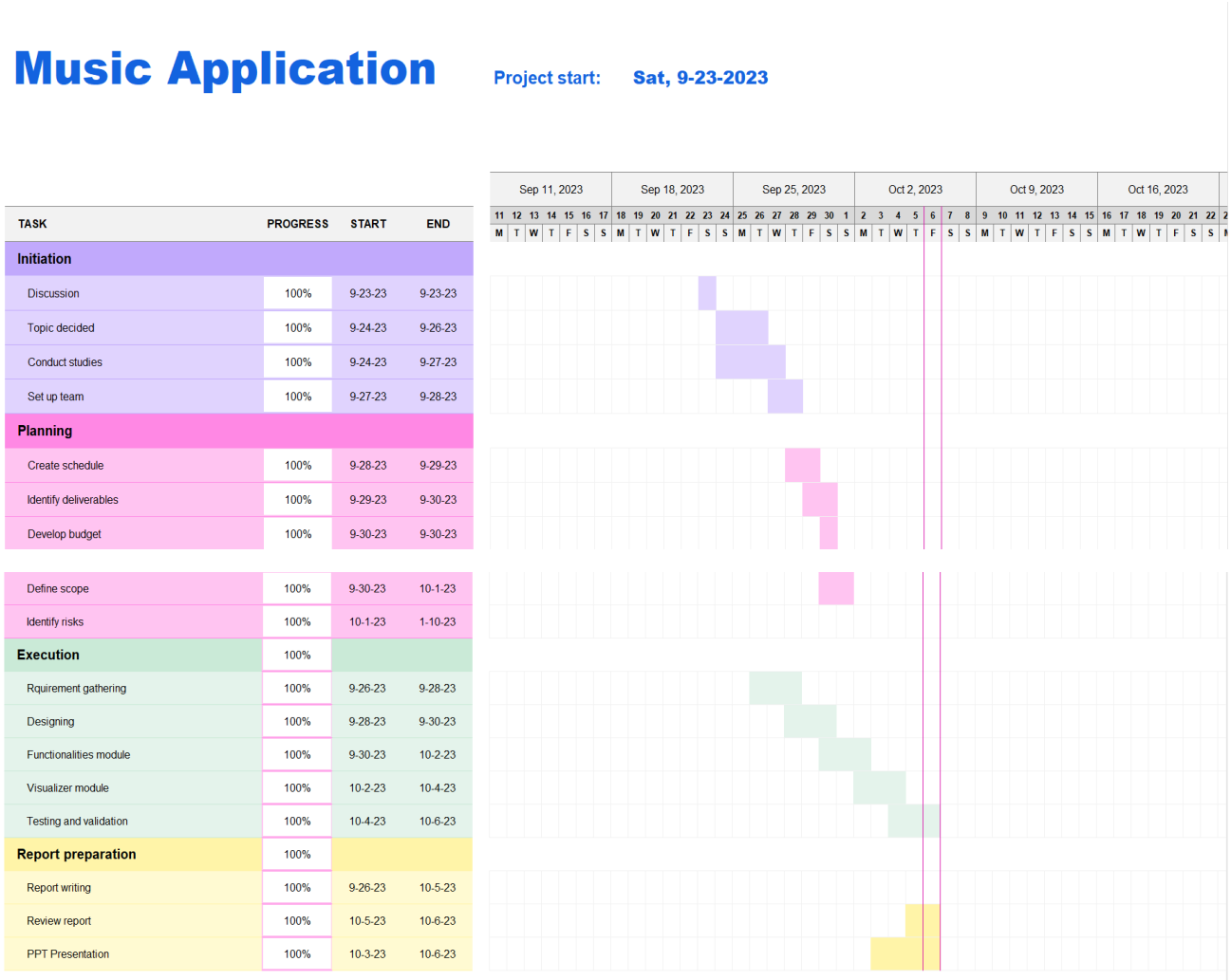
No known issues are listed at the time of this document's creation.

CHAPTER 3 – PROJECT PLANNING

In this project, we have used the “Gannt Chart” as the tool for Project Management.

A Gantt chart is a project management tool that illustrates work completed over a period of time in relation to the time planned for the work. It typically includes two sections: the left side outlines a list of tasks, while the right side has a timeline with schedule bars that visualize work.

It assists in the planning and scheduling of projects of all sizes; they are particularly useful for visualising projects. A Gantt chart is defined as a graphical representation of activity against time; it helps project professionals monitor progress.



TASK	PROGRESS	START	END
Initiation			
Discussion	100%	9-23-23	9-23-23
Topic decided	100%	9-24-23	9-26-23
Conduct studies	100%	9-24-23	9-27-23
Set up team	100%	9-27-23	9-28-23
Planning			
Create schedule	100%	9-28-23	9-29-23
Identify deliverables	100%	9-29-23	9-30-23
Develop budget	100%	9-30-23	9-30-23
Define scope	100%	9-30-23	10-1-23
Identify risks	100%	10-1-23	1-10-23
Execution	100%		
Rquirement gathering	100%	9-26-23	9-28-23
Designing	100%	9-28-23	9-30-23
Functionalities module	100%	9-30-23	10-2-23
Visualizer module	100%	10-2-23	10-4-23
Testing and validation	100%	10-4-23	10-6-23
Report preparation	100%		
Report writing	100%	9-26-23	10-5-23
Review report	100%	10-5-23	10-6-23
PPT Presentation	100%	10-3-23	10-6-23

FIG. – 3.1 - GANNT CHART OF MUSIC APPLICATION

CHAPTER 4 – IMPLEMENTATION

Code

```
import tkinter as tk

import fnmatch

import os

from pygame import mixer

from tkinter import Scale

import numpy as np

from matplotlib.figure import Figure

from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

import librosa.display



#updated code...

fig = Figure(figsize=(4, 2), dpi=100)

ax = fig.add_subplot(111)

canvas = tk.Tk()

canvas.title("Music Player")

canvas.geometry("600x800")

canvas.config(bg='white')

#updated code...
```

```
canvas_visualizer = FigureCanvasTkAgg(fig, master=canvas)

canvas_visualizer.get_tk_widget().pack(side=tk.BOTTOM, fill=tk.BOTH, expand=True)

rootpath = "C:\\Users\\Aryan Kumar\\Desktop\\Audioplayer\\music"

pattern = "*.mp3"

mixer.init()

prev_img = tk.PhotoImage(file = "C:\\Users\\Aryan Kumar\\Desktop\\Audioplayer\\prev_img.png")

stop_img = tk.PhotoImage(file = "C:\\Users\\Aryan Kumar\\Desktop\\Audioplayer\\stop_img.png")

power = tk.PhotoImage(file = "C:\\Users\\Aryan Kumar\\Desktop\\Audioplayer\\power.png")

play_img = tk.PhotoImage(file = "C:\\Users\\Aryan Kumar\\Desktop\\Audioplayer\\play_img.png")

pause_img = tk.PhotoImage(file = "C:\\Users\\Aryan Kumar\\Desktop\\Audioplayer\\pause_img.png")

next_img = tk.PhotoImage(file = "C:\\Users\\Aryan Kumar\\Desktop\\Audioplayer\\next_img.png")

def select():

    label.config(text = listBox.get("anchor"))

    mixer.music.load(rootpath + "\\" + listBox.get("anchor"))

    mixer.music.play()

def stop():

    mixer.music.stop()
```

```
    listBox.select_clear('active')
def play_next():
    next_song = listBox.curselection()
    next_song = next_song[0] + 1
    next_song_name = listBox.get(next_song)
    label.config(text = next_song_name)
    mixer.music.load(rootpath + "\\" + next_song_name)
    mixer.music.play()
    listBox.select_clear(0, 'end')
    listBox.activate(next_song)
    listBox.select_set(next_song)
def play_prev():
    next_song = listBox.curselection()
    next_song = next_song[0] - 1
    next_song_name = listBox.get(next_song)
    label.config(text = next_song_name)
    mixer.music.load(rootpath + "\\" + next_song_name)
    mixer.music.play()
    listBox.select_clear(0, 'end')
    listBox.activate(next_song)
    listBox.select_set(next_song)
```

```
def pause_song():
    mixer.music.pause()
def play_song():
    mixer.music.unpause()
def adjust_volume(volume):
    mixer.music.set_volume(float(volume))



def update_visualizer():
    try:
        if mixer.music.get_busy():
            # Get the current music position in seconds
            current_pos = mixer.music.get_pos() / 1000

            # Load the currently playing audio file
            audio_file = rootpath + "\\" + listBox.get("anchor")

            # Read the audio file and extract the waveform data
            y, sr = librosa.load(audio_file)
            duration = librosa.get_duration(y=y, sr=sr)
```

```
        # Calculate the desired plot length in seconds
        plot_length = 1.0  # Adjust this value as needed

        # Calculate the start and end time based on the current position and plot
length
        start_time = max(0, current_pos - plot_length)
        end_time = current_pos

        # Convert the start and end time to sample indices
        start_index = int(start_time * sr)
        end_index = int(end_time * sr)

        # Slice the waveform data
        data = y[start_index:end_index]

        # Clear the axis and plot the waveform data
        ax.clear()
        ax.plot(data, color='darkblue')

        # Adjust the width of the plot
        ax.set_xlim(0, len(data))
```

```
            ax.set_ylim(-1, 1)


            # Redraw the canvas
            canvas_visualizer.draw()


    except Exception as e:
        print(f"Error updating visualizer: {str(e)}")


    # Schedule the next update
    canvas.after(50, update_visualizer)


listBox = tk.Listbox(canvas, fg = "darkblue", bg = "lightgrey", width = 100, font = ('arial',14))
listBox.pack(padx = 15,pady = 15)
label = tk.Label(canvas, text = '', bg = 'white', fg = 'red', font = ('arial',18))
label.pack(pady = 15)
top = tk.Frame(canvas, bg = 'lightgrey')
top.pack(padx = 10, pady = 5, anchor = 'center')
prevButton = tk.Button(canvas, text = "Prev", image = prev_img, bg = 'lightgrey', borderwidth = 0, command = play_prev)
prevButton.pack(pady = 15, in_ = top, side = 'left')
```

```
stopButton = tk.Button(canvas, text = "Stop", image = stop_img, bg = 'lightgrey', borderwidth = 0, command = stop)

stopButton.pack(pady = 15, in_ = top, side = 'left')

powerButton = tk.Button(canvas, text = "Prev", image = power, bg = 'lightgrey', borderwidth = 0, command = select)

powerButton.pack(pady = 15, in_ = top, side = 'left')

playButton = tk.Button(canvas, text = "Prev", image = play_img, bg = 'lightgrey', borderwidth = 0, command = play_song)

playButton.pack(pady = 15, in_ = top, side = 'left')

pauseButton = tk.Button(canvas, text = "Stop", image = pause_img, bg = 'lightgrey', borderwidth = 0, command = pause_song)

pauseButton.pack(pady = 15, in_ = top, side = 'left')

nextButton = tk.Button(canvas, text = "Stop", image = next_img, bg = 'lightgrey', borderwidth = 0, command = play_next)

nextButton.pack(pady = 15, in_ = top, side = 'left')

volumeScale = Scale(canvas, from_=0, to=1, resolution=0.1, orient="horizontal", length=200, bg="white", fg="black", troughcolor="gray", sliderrelief="flat", command=adjust_volume)

volumeScale.pack(pady=15, in_=top, side='left')

for root, dirs, files, in os.walk(rootpath):

    for filename in fnmatch.filter(files, pattern):

        listBox.insert('end',filename)

update_visualizer()

canvas.mainloop()
```

OUTPUT

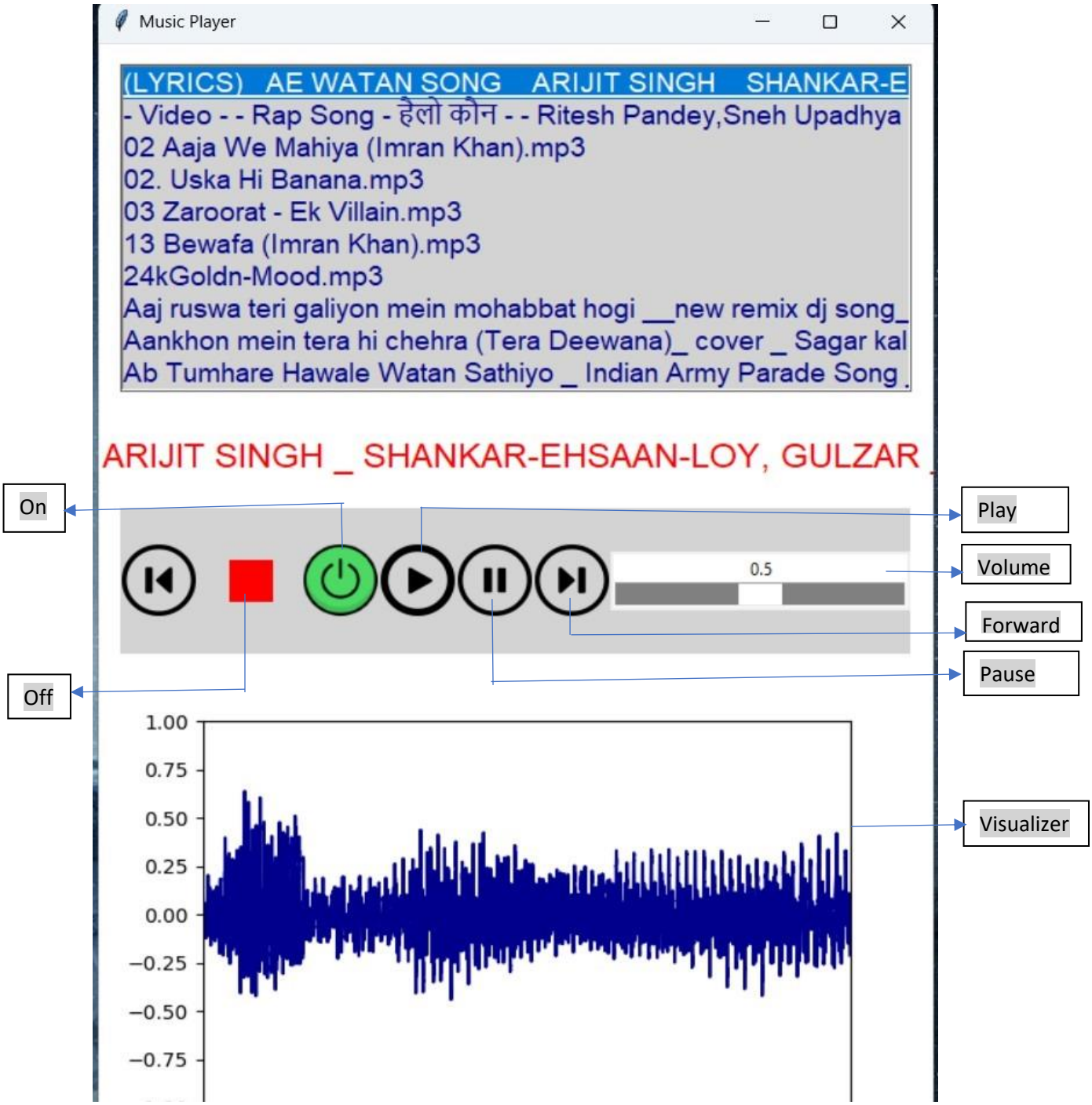


FIG. – 5.1

CHAPTER 5 – CONCLUSION

In conclusion, the development of a music application system represents a dynamic and innovative venture that can greatly enhance the way people interact with music in the digital age. This system offers a myriad of opportunities for both users and developers alike, with the potential to revolutionize the music industry and the way we consume and create music.

Through careful planning, user-centred design, and robust technology, such a system can provide a seamless and enjoyable music experience for users. It can empower artists to connect with their audience more effectively, help users discover new music, and create a vibrant community around music appreciation.

However, the success of a music application system depends on various factors, including its functionality, user-friendliness, content diversity, and continuous updates to keep up with evolving trends and technologies. Moreover, it must address copyright and licensing issues to ensure that artists and content creators are fairly compensated.

In summary, the development of a music application system holds immense potential for enriching our musical experiences and fostering greater engagement with music. With a commitment to user satisfaction, innovation, and legal compliance, such a system can play a significant role in shaping the future of the music industry.