Karnataka Law Society's

GOGTE INSTITUTE OF TECHNOLOGY

UDYAMBAG, BELAGAVI-590010

(An Autonomous Institution under Visvesvaraya Technological University, Belagavi)

(APPROVED BY AICTE, NEW DELHI)

Department of Computer Science and Engineering

A Project Report on

"MusicPlayer"

Submitted By:

Chandrakanth U K 2GI19CS406

Nihal M P 2GI19CS413

Shriniket K 2GI19CS421

Prinyank P 2GI19CS417

GUIDE BY: Mr.Gajendra D **HOD:Dr.Vijay.S.R**

2019– 2020

CERTIFICATE

This is to certify that the project entitled ***" MusicPlayer"*** *is abonafide record of the Project work done by,* ***Chandrakanth U K*** **(2GI19CS406)** *under my supervision and guidance, in partial fulfillment of the requirements for the Outcome Based Education Paradigm Computer science Engineering from Gogte Institute of Technology ,Belgaum for the academic year 2019-20*

Mr.Gajendra D
Asst. Professor

Dr.Vijay.S.R
Professor & Head
Dept.of Computer Science and Engg.

Place: KLS Gogte Institute of Technology, Belgaum.
Date:22-12-2020

ACKNOWLEDGEMENT

This group feels greatly indebted to Computer Science and Engineering Department, for the opportunity given us to undertake the "**MusicPlayer**" project. This project includes thoughts and contribution of many individuals. And we wish to express our sincere appreciation and gratitude to them.

First and foremost we want to extend entire your gratitude to our lecturer **Mr.Gajendra D** for sharing her/his knowledge and profound wisdom with us. We appreciate all his comments and suggestions, which are incorporated into this project.

We would also like to express our gratitude toward group members. Without their help, support, and encouragement, this project would never had been completed.

In our respect, this project is an outcome of the learning experience we have shared with our fellow students. We dedicate this project to all our fellow engineering students.

Group member names:

Chandrakanth U K **2GI19CS406**

Course Project Report and PPT Content

1. Title
2. Problem statement for that the project
3. Objectives of Defined Problem statement
4. Design / Algorithm/Flowchart/Methodology
5. Implementation details/Function/Procedures/Classes and Objects (Language/Tools)
6. Working model of the final solution
7. Report and Oral Presentation skill

Marks allocation:

	Batch No. :					
1.	Project Title:	Marks Range	USN			
2.	Problem statement (PO2)	0-1				
3.	Objectives of Defined Problem statement (PO1,PO2)	0-2				
4.	Design / Algorithm/Flowchart/Methodology (PO3)	0-3				
5.	Implementation details/Function/Procedures/Classes and Objects (Language/Tools) (PO1,PO3,PO4,PO5)	0-4				
6.	Working model of the final solution (PO3,PO12)	0-5				
7.	Report and Oral presentation skill (PO9,PO10)	0-5				
	Total	20				

*** 20 marks is converted to 10 marks for CGPA calculation**

1.Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.

2.Problem Analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and Engineering sciences.

3.Design/Development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4.Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5.Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6.The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7.Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need

for sustainable development.

8.Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9.Individual and team work: Function effectively as an individual and as a member or leader in diverse teams, and in multidisciplinary settings.

10.Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.

Abstract

The **MusicPlayer** is very useful for playing audio files, this project is designed and implemented using Pyqt library with a simple UI and limited features. The goal of this project is to develop and implement a simple music player with an easy to use GUI's, the application only supports mp3 file and consumes less systems resources.

The static structure of the application is represented by the Class diagram and the working of the system and interaction with user as well as flow of the system is represented using Usecase , Scenario,Sequence and Activity diagrams.

Table of Contents

8.Advantages

9. Limitation

10. .Conclusions

1. Introduction

(i) Purpose of This Document

The requisite details of the Program MusicPlayer are included in this report document. You can understand the principles, architecture and software specifications and be familiar with the way our project operates as well.

(ii) Scope of this Product:

- The project is basically GUI based system and provides good user interaction.
- Here we show the name of the Musicfile.
- Here we load the available Music folder s.
- A user can also be able to see the information on the file that is playing.
- feature that allows users to repeat currently playing files or even shuffle the list of files to be played.
- Drak and light themes are available

(iii) Overview:

This project system gives better option for playing .mp3 formated audio files and with satisfied user interface . software need minimum requirements for good excution and so no need of maintenance costs**.**dark and lights themes supports goode UI's.

2. Performance Requirements

(i)Software Requirements

No need of any software's requirements for run because software is in form of *.exe* format either you need to run by code on editor need followings

- Runtime Environment - python 3,Anaconda
- Operating system - windows 7,8 & 10

(ii)Hardware Requirement

- Professor - 2.4 GHz Processor or more
- Memory - 2 GB RAM or more
- Disk space -10 GB or more

3. Product Interfaces

(i)System interfaces

The application runs in the latest version of Windows and Linux meanly which OS support Python Environments

(ii)User interfaces

The application GUI provides Menus, Toolbars, Buttons, Seekbar, Containers, Grids allowing for easy control by a keyboard and a mouse.

4. Product Functions:

(i)Functional Attributes

- This MP3 software that allows it's users to play .mp3 file
- Pause and Resumes options are provides
- User can put music in shuffle or in loop
- In this software allows user's to fast forward or rewind it.
- User can set Playlist how they wants to.
- Add file by drag and drop or by browsing in Pcs

(ii)Non-Functional Attributes

In this, non-functional attributes are explained that are required by software system for better performance which are also known as *quality attributes*..

5.Technology used:

Here we using python for developing the desktop software for better feasible and flexibility to User. Following are packages and liberies used

- **PyQt** is package used in python for developing GUI base software. It is more efficient than other packages like TKinter.
- In this package provides cross-platform, dependency-free audio playback capability for Python 3 on Windows and Linux.

6.UML Diagrams

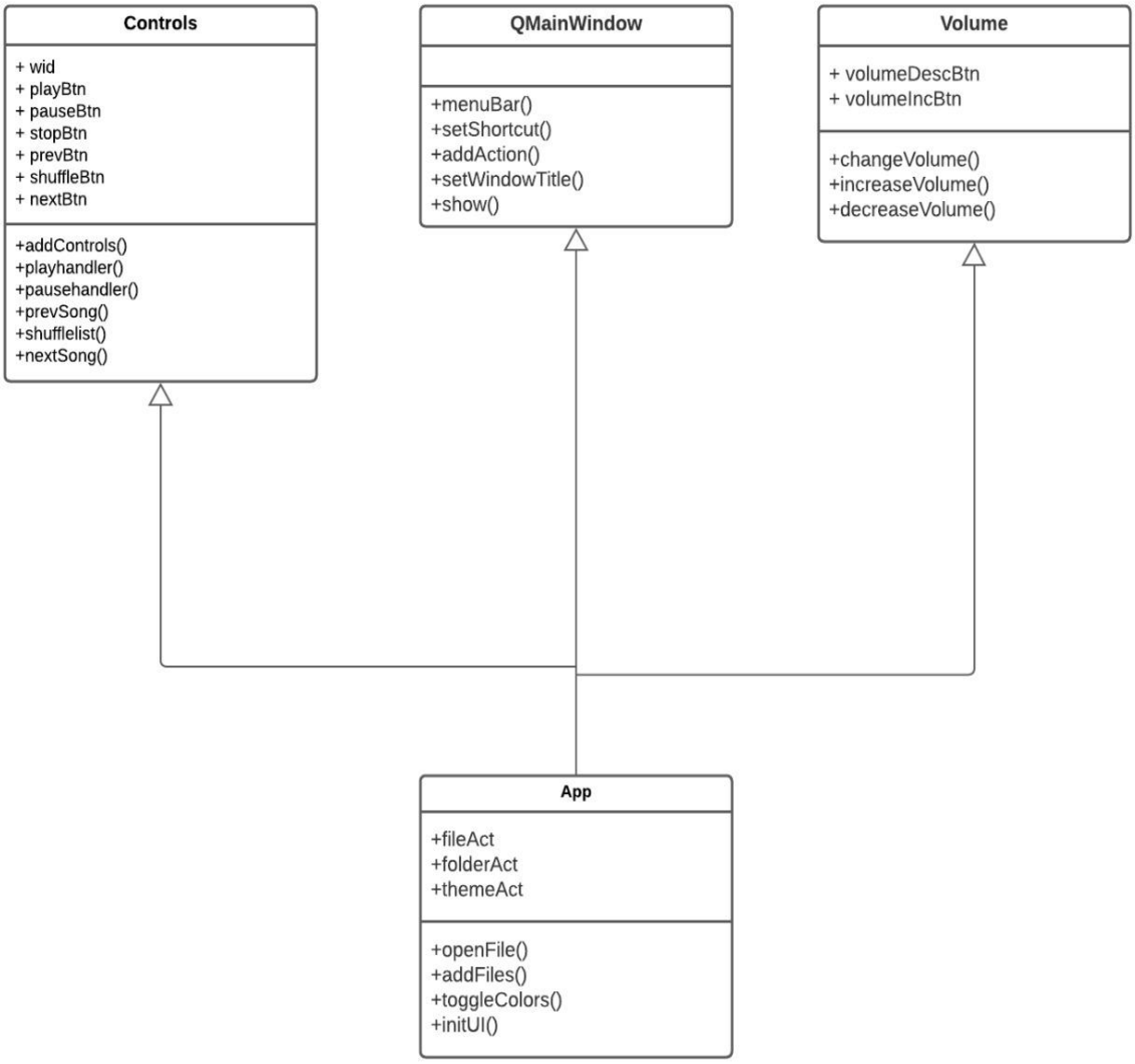


Figure 1.1 Class diagram of Music Player

Use Cases:

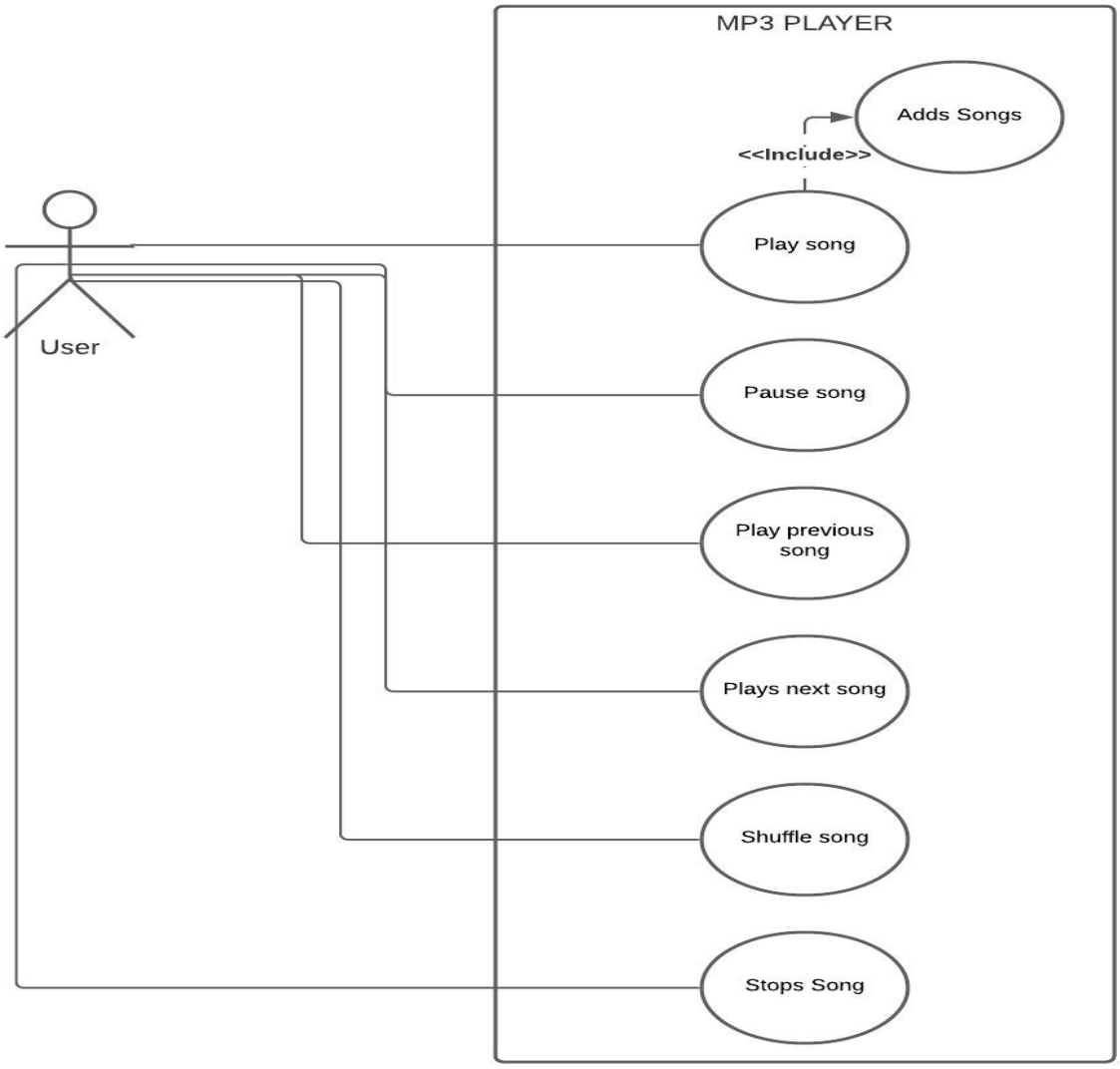


Figure 1.2 Use Case diagram of Music Player

Scenario :

User launche the MP3 player
Application loads up and get ready to use
User adds the MP3 File
MP3 Player loads the corresponding file and starts playing the song
User can pause the song
MP3 Player responds by pausing the song
User can skip the song and as well as stop the song
MP3 Player responds by applying the corresponding actions requested by the user
User terminates the application

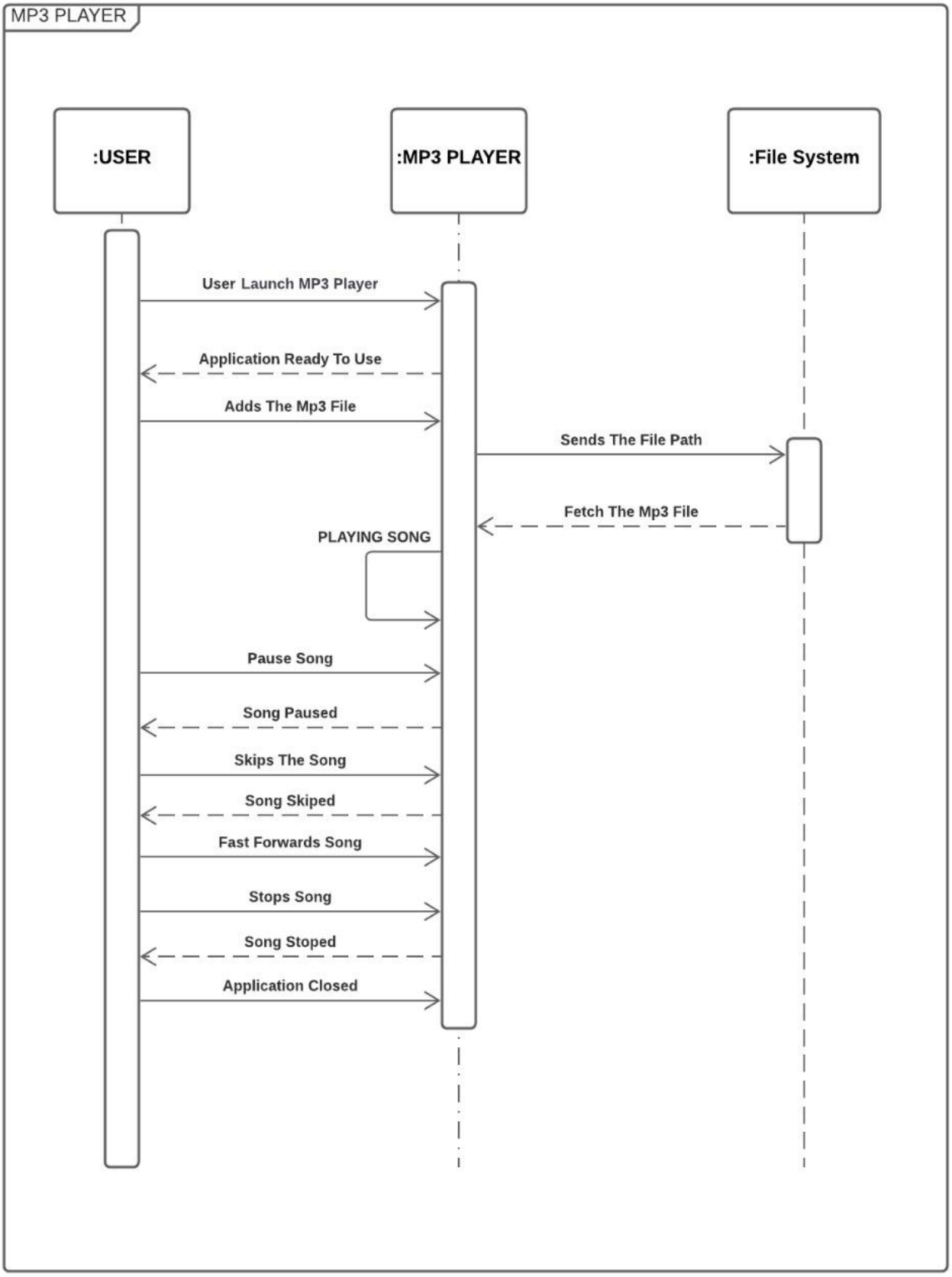


Figure 1.3 Sequence diagram of Music Player

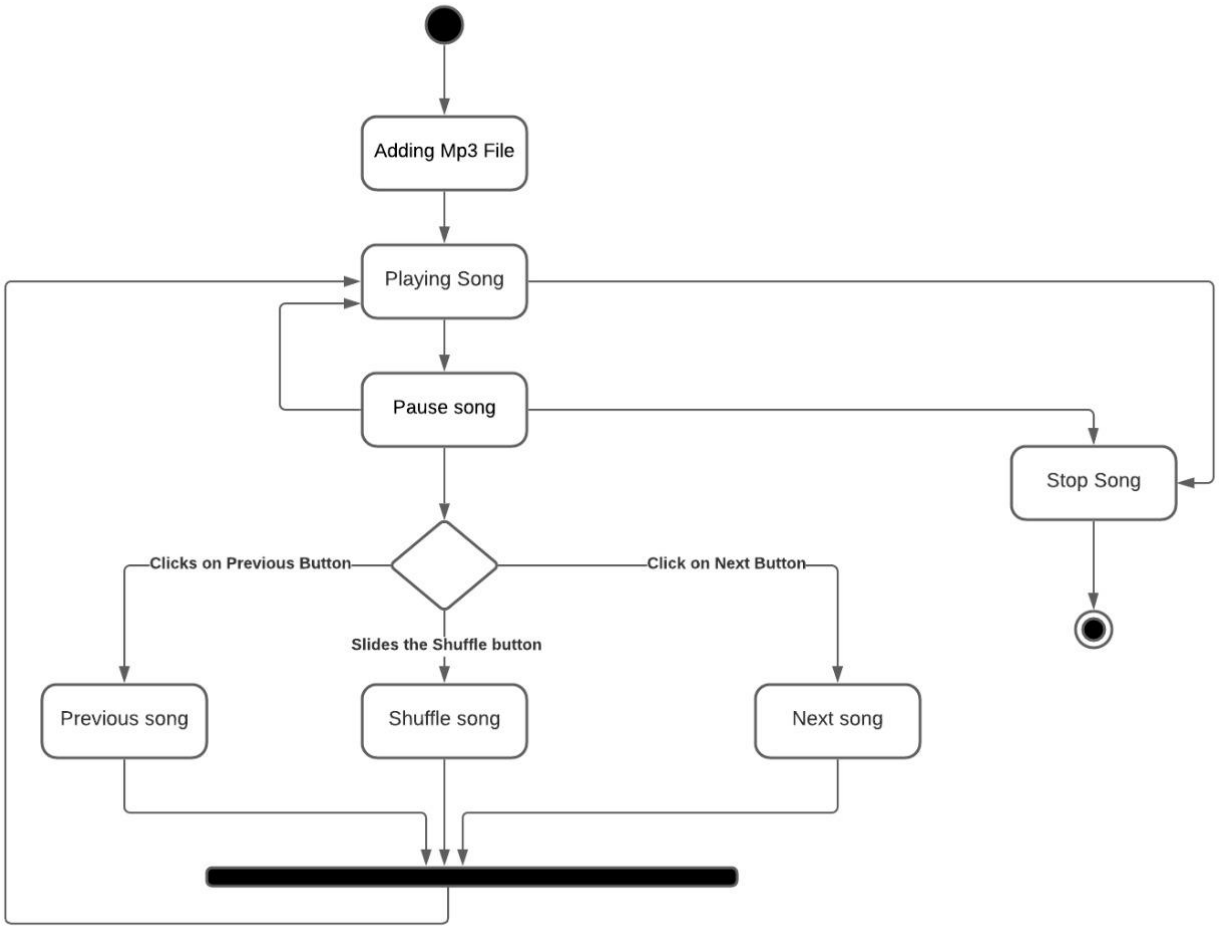


Figure 1.4 Activity diagram of Music Player

7.Development:

Code:

```
import sys
from PyQt5.QtGui import QPalette, QColor, QIcon
from PyQt5.QtCore import QUrl, QDirIterator, Qt
from PyQt5.QtWidgets import QApplication, QWidget, QMainWindow, QPushButton, QFileDialog, QAction, QHBoxLayout, QVBoxLayout, QSlider
from PyQt5.QtMultimedia import QMediaPlaylist, QMediaPlayer, QMediaContent


class Controls():
    def __init__(self):
        pass



    def addControls(self):
        wid = QWidget(self)
        self.setCentralWidget(wid)
        # Add song controls
        self.Slider = QSlider(Qt.Horizontal)
        self.Slider.setFocusPolicy(Qt.NoFocus)
        self.Slider.setRange(0, 0)
        self.Slider.sliderMoved.connect(self.set_position)

        playBtn = QPushButton('Play')  # play button
        pauseBtn = QPushButton('Pause')  # pause button
        stopBtn = QPushButton('Stop')  # stop button
        # Add playlist controls
        volumeDescBtn = QPushButton('V (-)')  # Decrease Volume
        volumeIncBtn = QPushButton('V (+)')  # Increase Volume

        prevBtn = QPushButton('Prev')
        shuffleBtn = QPushButton('Shuffle')
        nextBtn = QPushButton('Next')
        # Add button layouts
        controlArea = QVBoxLayout()  # centralWidget
        controls = QHBoxLayout()
        playlistCtrlLayout = QHBoxLayout()
        # Add buttons to song controls layout

        controls.addWidget(volumeDescBtn)
        controls.addWidget(prevBtn)
        controls.addWidget(shuffleBtn)
        controls.addWidget(nextBtn)
        # Add buttons to playlist controls layout
        controls.addWidget(volumeIncBtn)
        playlistCtrlLayout.addWidget(playBtn)
```

```
        playlistCtrlLayout.addWidget(pauseBtn)
        playlistCtrlLayout.addWidget(stopBtn)

        # Add to vertical layout

        controlArea.addLayout(controls)
        controlArea.addLayout(playlistCtrlLayout)
        controlArea.addWidget(self.Slider)
        wid.setLayout(controlArea)

        # Connect each signal to their appropriate function
        playBtn.clicked.connect(self.playhandler)
        pauseBtn.clicked.connect(self.pausehandler)
        stopBtn.clicked.connect(self.stophandler)
        volumeDescBtn.clicked.connect(self.decreaseVolume)
        volumeIncBtn.clicked.connect(self.increaseVolume)

        prevBtn.clicked.connect(self.prevSong)
        shuffleBtn.clicked.connect(self.shufflelist)
        nextBtn.clicked.connect(self.nextSong)

        self.statusBar()
        self.playlist.currentMediaChanged.connect(self.songChanged)
        self.player.positionChanged.connect(self.position_change)
        self.player.durationChanged.connect(self.duration_change)

    def openFile(self):
        song = QFileDialog.getOpenFileName(self, "Open Song", "~", "Sound Files (*.mp3 *.ogg
*.wav *.m4a)")

        if song[0] != '':
            url = QUrl.fromLocalFile(song[0])
            if self.playlist.mediaCount() == 0:
                self.playlist.addMedia(QMediaContent(url))
                self.player.setPlaylist(self.playlist)
                self.player.play()
                self.userAction = 1
            else:
                self.playlist.addMedia(QMediaContent(url))

    def addFiles(self):
        if self.playlist.mediaCount() != 0:
            self.folderIterator()
        else:
            self.folderIterator()
            self.player.setPlaylist(self.playlist)
            self.player.playlist().setCurrentIndex(0)
            self.player.play()
            self.userAction = 1

    def folderIterator(self):
        folderChosen = QFileDialog.getExistingDirectory(self, 'Open Music Folder', '~')
        if folderChosen != None:
```

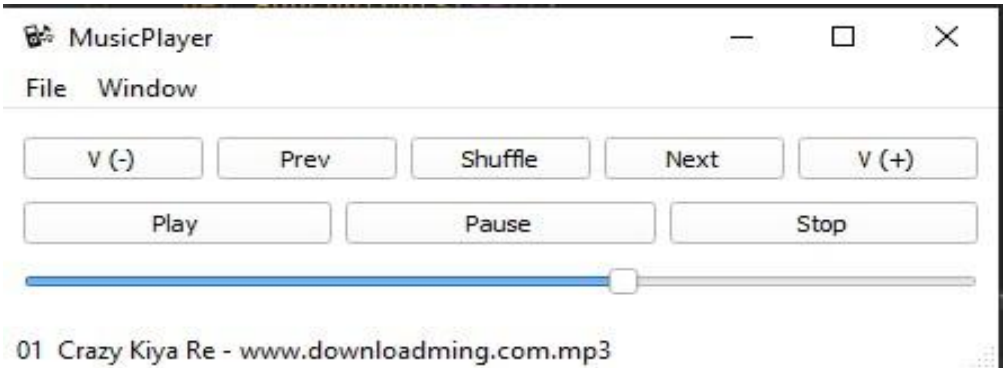
```
        it = QDirIterator(folderChosen)
        it.next()
        while it.hasNext():
            if it.fileInfo().isDir() == False and it.filePath() != '.':
                fInfo = it.fileInfo()
                if fInfo.suffix() in ('mp3', 'ogg', 'wav', 'm4a'):
                    self.playlist.addMedia(QMediaContent(QUrl.fromLocalFile(it.filePath())))
            it.next()
        if it.fileInfo().isDir() == False and it.filePath() != '.':
            fInfo = it.fileInfo()
            if fInfo.suffix() in ('mp3', 'ogg', 'wav', 'm4a'):
                self.playlist.addMedia(QMediaContent(QUrl.fromLocalFile(it.filePath())))

    def playhandler(self):
        if self.playlist.mediaCount() == 0:
            self.openFile()
        elif self.playlist.mediaCount() != 0:
            self.player.play()
            self.userAction = 1

    def pausehandler(self):
        self.userAction = 2
        self.player.pause()

    def stophandler(self):
        self.userAction = 0
        self.player.stop()
        self.playlist.clear()
        self.statusBar().showMessage("Stopped and cleared playlist")



    def prevSong(self):
        if self.playlist.mediaCount() == 0:
            self.openFile()
        elif self.playlist.mediaCount() != 0:
            self.player.playlist().previous()

    def shufflelist(self):
        self.playlist.shuffle()

    def nextSong(self):
        if self.playlist.mediaCount() == 0:
            self.openFile()
        elif self.playlist.mediaCount() != 0:
            self.player.playlist().next()

    def songChanged(self, media):
        if not media.isNull():
            url = media.canonicalUrl()
            self.statusBar().showMessage(url.fileName())

    def position_change(self, position):
```

```
        self.Slider.setValue(position)

    def duration_change(self, duration):
        self.Slider.setRange(0, duration)

    def set_position(self, position):
        self.player.setPosition(position)

    def toggleColors(self):

        app.setStyle("Fusion")
        palette = QPalette()
        if self.color == 0:
            palette.setColor(QPalette.Window, QColor(53, 53, 53))
            palette.setColor(QPalette.WindowText, Qt.white)
            palette.setColor(QPalette.Base, QColor(25, 25, 25))
            palette.setColor(QPalette.AlternateBase, QColor(53, 53, 53))
            palette.setColor(QPalette.ToolTipBase, Qt.white)
            palette.setColor(QPalette.ToolTipText, Qt.white)
            palette.setColor(QPalette.Text, Qt.white)
            palette.setColor(QPalette.Button, QColor(53, 53, 53))
            palette.setColor(QPalette.ButtonText, Qt.white)
            palette.setColor(QPalette.BrightText, Qt.red)
            palette.setColor(QPalette.Link, QColor(235, 101, 54))
            palette.setColor(QPalette.Highlight, QColor(235, 101, 54))
            palette.setColor(QPalette.HighlightedText, Qt.black)
            app.setPalette(palette)
            self.color = 1
        elif self.color == 1:
            palette.setColor(QPalette.Window, Qt.white)
            palette.setColor(QPalette.WindowText, Qt.black)
            palette.setColor(QPalette.Base, QColor(240, 240, 240))
            palette.setColor(QPalette.AlternateBase, Qt.white)
            palette.setColor(QPalette.ToolTipBase, Qt.white)
            palette.setColor(QPalette.ToolTipText, Qt.white)
            palette.setColor(QPalette.Text, Qt.black)
            palette.setColor(QPalette.Button, Qt.white)
            palette.setColor(QPalette.ButtonText, Qt.black)
            palette.setColor(QPalette.BrightText, Qt.red)
            palette.setColor(QPalette.Link, QColor(66, 155, 248))
            palette.setColor(QPalette.Highlight, QColor(66, 155, 248))
            palette.setColor(QPalette.HighlightedText, Qt.black)
            app.setPalette(palette)
            self.color = 0

class volume():
    def __init__(self):
        pass

    def changeVolume(self, value):
        self.player.setVolume(value)

    def increaseVolume(self):
```

```
        vol = self.player.volume()
        vol = min(vol + 5, 100)
        self.player.setVolume(vol)

    def decreaseVolume(self):
        vol = self.player.volume()
        vol = max(vol - 5, 0)
        self.player.setVolume(vol)

class App(QMainWindow,Controls,volume):

    def __init__(self):
        super().__init__()
        super(App, self).__init__()
        self.player = QMediaPlayer()
        self.playlist = QMediaPlaylist()
        self.title = 'MusicPlayer'
        self.left = 300
        self.top = 300
        self.width = 300
        self.height = 150
        self.color = 0  # 0- toggle to dark 1- toggle to light
        self.userAction = -1  # 0- stopped, 1- playing 2-paused
        self.initUI()

    def initUI(self):
        # Add file menu
        menubar = self.menuBar()
        filemenu = menubar.addMenu('File')
        windowmenu = menubar.addMenu('Window')
        self.setWindowIcon(QIcon('logo1.ico'))

        fileAct = QAction('Open File', self)
        folderAct = QAction('Open Folder', self)
        themeAct = QAction('Toggle light/dark theme', self)

        fileAct.setShortcut('Ctrl+O')
        folderAct.setShortcut('Ctrl+D')
        themeAct.setShortcut('Ctrl+T')

        filemenu.addAction(fileAct)
        filemenu.addAction(folderAct)
        windowmenu.addAction(themeAct)

        fileAct.triggered.connect(self.openFile)
        folderAct.triggered.connect(self.addFiles)
        themeAct.triggered.connect(self.toggleColors)

        self.addControls()

        self.setWindowTitle(self.title)
        self.setGeometry(self.left, self.top, self.width, self.height)
        self.toggleColors()
```

```
        self.show()


if __name__ == '__main__':
    app = QApplication(sys.argv)
    ##ex = App()
    sys.exit(app.exec_())
```

Outputs:



Demonstration by using both dark and light themes

8.Advantage:

- Took less memory space
- Users can load full music folder to application
- Supports Drak and light themes
- Shuffle the music list
- User friendly GUI

9.Limitation:

- contains only some limited features
- Non – applied Internet Access

10.Conclusion :

The goal is to build an MP3 software that allows it's users to play MP3 .we need to focus on building a beautiful user interface for the better engagement of the users. The interface will be listing the MP3 files

References:

Book's:

[1] Michael Urban and Joel Murach , Python Programming, Murach,2016

Site's:

[1] https://www.geeksforgeeks.org

[2] https://realpython.com/playing-and-recording-sound-python/