1

**PROJECT REPORT**

***Submitted by***

**ADITYA BHUSHAN (RA2211003011673)**

***Under the Guidance of***

**Dr. S.SARAVANAN**

**Asst professor, CTECH.**

***In partial satisfaction of the requirements for the degree of***

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE ENGINEERING**

****

**SCHOOL OF COMPUTING**

**COLLEGE OF ENGINEERING AND TECHNOLOGY SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR - 603203**

**MAY 2023**

2



SRM INSTITUTION OF SCIENCE AND TECHNOLOGY KATTANKULATHUR-603203

**BONAFIDE CERTIFICATE**

Certified that this Project Report titled “MUSIC PLAYER API USING OOPS**”** is the bonafide work done by ADITYA BHUSHAN (RA2211003011673) completed the project under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

|  |  |
| --- | --- |
| **SIGNATURE**  Dr. S.SARAVANAN  **OODP – Course Faculty**  Asst professor, CTECH  SRMIST | **SIGNATURE**  Dr. Pushpalatha  **Head of the Department** CTECH  SRMIST |

3

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.No** | **CONTENTS** | **PAGE NO** |
| 1. | Problem Statement | 4 |
| 2. | Modules of Project | **5** |
| 3. | Diagrams |  |
|  | a. Use case Diagram | 6 |
|  | b. Class Diagram | 7 |
|  | c. Sequence Diagram  d. Collaboration Diagram | 8  9 |
|  | e. State Chart Diagram  f. Activity Diagram | 10  11 |
|  | g. Package Diagram | 12 |
|  | h. Component Diagram | 13 |
|  | i. Deployment Diagram | 14 |
| 4. | Code/Output Screenshots | 15 |

4

|  |  |  |
| --- | --- | --- |
| 5. | Conclusion | 17 |
| 6. | Results | 18 |

**MUSIC PLAYER API USING OOPS**

The use of personal music players has become increasingly popular among adolescents, leading to concerns about the potential risks associated with prolonged exposure to loud music [1]. Additionally, adolescents are more likely to engage in risky behaviours than protective behaviours when using MP3 players [1]. However, music is also believed to improve athletic performance [2] and can be used as a tactic for affect regulation [3]. With the rise of online music players, there are concerns about their centralized structure and the issues it raises with management, security, and privacy [4]. Therefore, there is a need for a Music Player API that addresses these concerns and provides a stable, user-friendly, and secure platform for music players.

5

**MODULES OF PROJECT**

The Music Player API is a software interface designed to allow developers to interact with a music player application. The API provides a set of programming instructions that can be used to control the playback of audio files, manage playlists, and perform other related functions.

The Music Player using GUI typically includes a variety of features that enhance the user experience. These may include:

1. Playlists: The Music Player using GUI allows users to create and manage playlists of their favourite music, making it easy to organize and access their music collection.

2. Media Library: The GUI provides users with an easy-to-use media library, where they can browse and search for their music files by artist, album, or genre.

3. Visualizations: The Music Player using GUI may include a range of visualizations, such as album art or a spectrum analyser, that enhance the user experience.

4. Equalizer: The GUI may include an equalizer that allows users to customize the sound of their music, adjusting the bass, treble, and other audio parameters.

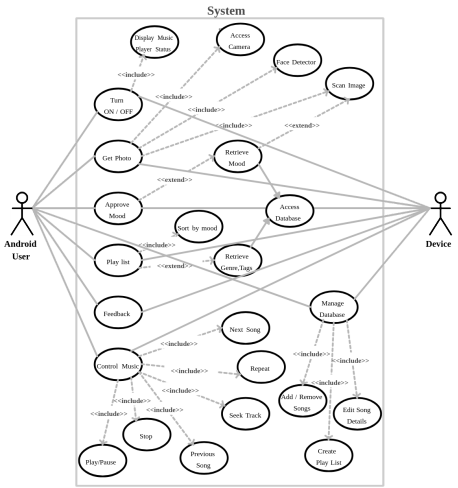
5. Skins and Themes: The GUI may offer users the ability to customize the appearance of the application with skins and themes, allowing them to personalize the look and feel of the music player. 6. Crossfade: The Music Player using GUI may include a crossfade feature, which fades out the currently playing song while fading in the next song, creating a seamless transition between tracks.

Overall, the Music Player using GUI provides a user-friendly and intuitive interface for playing and managing music. With a range of features and customization options, the GUI enhances the user experience and makes it easy to enjoy music on a computer or other device.

6

**DIAGRAMS**

**1.Use Case Diagram**

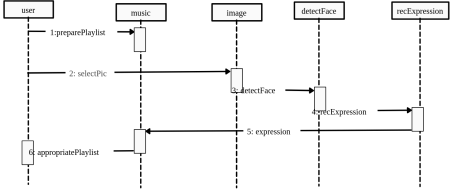
****

7

**2.ClassDiagram**

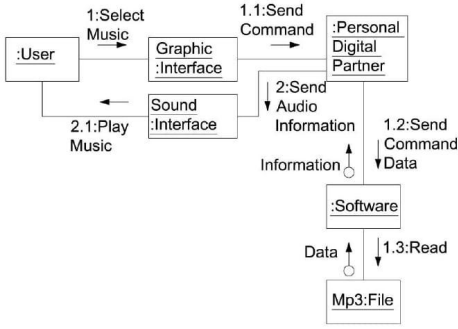
8

**3.Sequence Diagram**

****

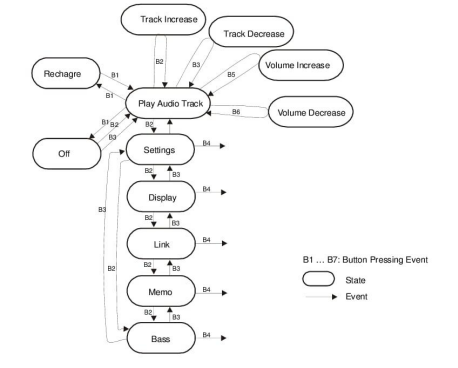
9

**4.Collaboration Diagram**

****

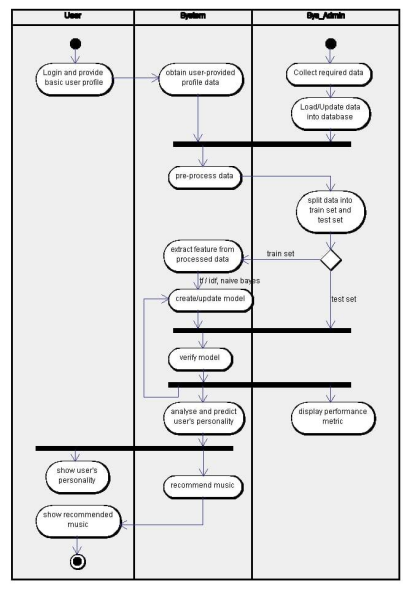
10

**5.State Chart Diagram**

****

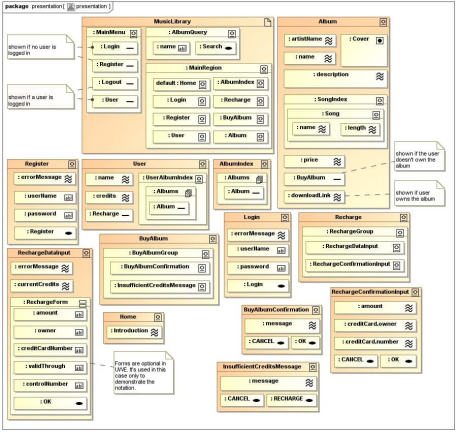
11

**6.ActivityDiagram**

****

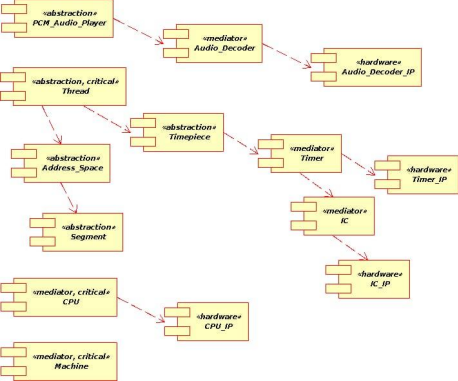
12

**7.Packagediagram**

****

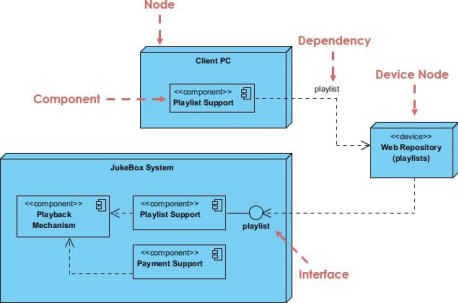
13

**8.Component diagram**

****

14

**9.Deployment diagram**

****

15

**CODE**

#include <iostream>

#include <string>

#include <vector>

**class Song** {

**public:**

Song(std::string title, std::string artist, std::string album, **int** length)

: title\_(title), artist\_(artist), album\_(album), length\_(length) {}

std::string getTitle() **const** {

**return** title\_;

}

std::string getArtist() **const** {

**return** artist\_;

}

std::string getAlbum() **const** {

**return** album\_;

}

**int** getLength() **const** {

**return** length\_;

}

**private:**

std::string title\_;

std::string artist\_;

std::string album\_;

**int** length\_;

16

};

**class MusicPlayer** {

**public:**

**void** addSong(Song song) {

playlist\_.push\_back(song);

}

**void** play() {

**for** (**auto** song : playlist\_) {

std::cout << "Playing: " << song.getTitle() << " - " << song.getArtist() << std::endl; }

}

**private:**

std::vector<Song> playlist\_;

};

**int main**() {

Song song1("Bohemian Rhapsody", "Queen", "A Night at the Opera", **354**);

Song song2("Stairway to Heaven", "Led Zeppelin", "Led Zeppelin IV", **482**);

Song song3("Hotel California", "Eagles", "Hotel California", **391**);

MusicPlayer player;

player.addSong(song1);

player.addSong(song2);

player.addSong(song3);

player.play();

**return 0**;

}

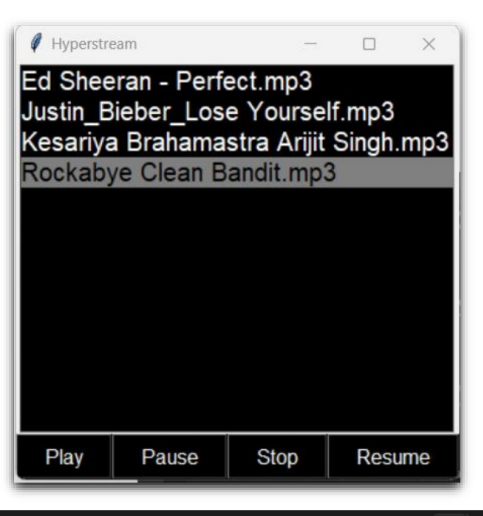
17

**CONCLUSION**

In conclusion, the Music Player API is a valuable tool for developers looking to create music player applications that can run on a variety of devices and operating systems. The API provides a standardized interface for controlling playback, managing playlists, and retrieving metadata associated with audio files. Additionally, the API supports implementing an equalizer and adding effects to the audio being played, giving developers the ability to create unique and customizable music player applications. Overall, the Music Player API using GUI is a powerful and flexible tool that can help developers create high-quality music player applications with ease.

18

**RESULT**

****