Project Report

On

Simple Music Player

(A Desktop Application)

Submitted by

Indu C	R170470
Madhu Priya R	R170340
Vinod Kumar N	R171004
Harsha Raj T	R171090

Under the guidance of

E Sushmitha
Asst. Professor

Department of Computer Science and Engineering

Rajiv Gandhi University of Knowledge Technologies
RK Valley, Kadapa (Dist), Andhra Pradesh, 516330.

Rajiv Gandhi University of Knowledge and Technologies(RGUKT), R.K.Valley, Kadapa, Andhra Pradesh.

CERTIFICATE

This is to certify that the project work titled "*Simple Music Player*" is a bonafied project work submitted by *Vinod Kumar N*, *Harsha Raj T*, *Indu C*, *Madhu Priya R* in the department of COMPUTER SCIENCE AND ENGINEERING in partial fulfillment of requirements for the award of degree of Bachelor of Technology in Computer science and engineering for the year 2020-2021 carried out the work under the supervision.

GUIDE
E SUSHMITHA

HEAD OF THE DEPARTMENT
P HARINADHA

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant guidance and encouragement crown all the efforts success.

I am extremely grateful to our respected Director, Prof. K. SANDHYA RANI for fostering an excellent academic climate in our institution.

I also express my sincere gratitude to our respected Head of the Department Mr. P HARINADHA for his encouragement, overall guidance in viewing this project a good asset and effort in bringing out this project.

I would like to convey thanks to our guide at college Ms. E SUSHMITHA for his guidance, encouragement, co-operation and kindness during the entire duration of the course andacademics.

My sincere thanks to all the members who helped me directly and indirectly in the completion of project work. I express my profound gratitude to all our friends and family members for their encouragement.

INDEX

Table of Contents

ABSTRACT

In the recent days it is scientifically proven that music is best stress reliever for many people irrespective of age. The function of playing music become essential in any device which consists of JDK environment. It is very convenient, but it contains controversial arguments about sound quality, So many users use music player applications.

By using these music applications people start thinking about relationship between music playing and sound quality. However those applications are not perfect, so it is hard to choose good application. This thesis is about the advantages of sound quality and the music player.

So we planned to suggest a music player application system design by analyzing applications by covering disadvantages of all other applications in the market.

The software is an updated version to overcome the problems that have occurred with online music players. The software provides easy access to play the music from local files. It will contain user friendly functions with attractive interfaces. With music, the player users can play/pause/stop a track. It is a fully functional music player like any other music player currently available.

Introduction

The software requirement specification mainly describes both functional and nonfunctional requirements for offline MP3 music players. This is mainly designed for providing offline music that you have on your computer locally. It is a free open-source program. The document also provides details of the external interfaces, performance considerations, and design constraints. This will play music properly without interference from advertisements.

Project Scope

The main object of this software is to play music with add free offline. In online music, time will be wasted because of the ads and also for the music selection. For that, we developed this project if we select a path the music stored in that path will be played automatically in a queue. The software includes next, previous, operations. An online music player has to maintain internet connectivity but in this software no need for internet. It uses the database for getting the data of all music present in the user's local library.

Advantages:

- The system allows you to play offline music.
- This application doesn't require any internet connection.
- This software is easy to use and has user friendly interface.
- Automatically it runs continuously according to the parent file.
- It is a freeware, portable and reliable

Disadvantages:

- Up to now it only supports mp3 files.
- Files must be pre downloaded.

Intended audience and reading suggestions

The intended audience of this document would be developers, project managers, users, and testers. Anyone with a programming background and some experience with UML can understand his document. The SRS document can be used in any case regarding the requirements of the project and the solutions that have been taken.
Here is a brief overview of the document.

1. Overall description of the project
2. External interfaces requirements
3. System features
4. Other non-functional requirements

OVERALL DESCRIPTION

Product perspective

The software is an updated version to overcome the problems that have occurred with online music players. The software provides easy access to play the music from local files. It will contain user-friendly functions with attractive interfaces. With music, the player users can play/pause/stop a track. It is a fully functional music player like any other music player currently available. The list of songs will be saved in a playlist after a user opens a file and the users can also select a song from that playlist that which user wants to listen to.

Product features

By using this app, the user can play tracks available in the offline library. Users can perform the following actions:

1) Users can play Mp3 music

2) Users can change the song by selecting the pause, previous, and next buttons.

3) Users can add songs to the playlist

4) The user has an access to manage the progress bar.

5) Simple interface

User classes and characteristics

There are mainly 3 users are there:

1) Naive user

2) Sophisticated user

3) Expert

One end-user class will be naive users, who will not know the internal architecture of the mp3 player; they can only use the application.

Other user classes can be sophisticated users having some knowledge of the software. They have some knowledge of how to insert, delete, and update software.

Other user classes can be of experts which will have full knowledge of hardware and software as well as their interfaces. The user should know the basics of how to operate the music player. The users with poor internet connectivity will benefit more.

Operating Environment

System requirements:

Operating System should be capable of playing music and have any of mentioned OS installed or installed "JDK" is enough to run the software.

Operating System:

It supports Microsoft Windows like Windows 7, Windows XP, Windows Vista, Mac OS, and Linux. Simply it can work the OS which are capable of running java programs.

Design and Implementation constraints

Software application needs to have an operating system that has enough performance. If the operating system does not have enough hardware resources available for the application. Then, there may be scenarios where the application does not work as intended or even at all. Simply all it need is installed “JDK” in the system with any type of OS since java is platform independent.

While deploying the software at different platforms, the user must have to install all SDK files. At some point user may face volume issues on switching audio files which has rectified. In order to run jar file of this project system JDK should consist of java SDK or else user has to download it manually and also has to specify the path in the VM configurations of the Project File.

Assumptions and Dependencies

Some software uses high cost for implementing the system and the client also agreed to that. It is assumed that the client won’t change the decision in the next phases. We used various online open-source materials for most of our project work. We integrated various components from other projects to make the application work as a whole.

Our software or project doesn’t need high level configurations it just need java setup or environment to run with this it can able to run in any OS. By this we can achieve compatibility also.

To complete this project I have used “javafx-graphics-11-linux.jar” file to implement media class and its properties which allows us to run any media file with proper code.

SYSTEM FEATURES

Functional Requirements

Open file:

When the user opens the application, it shows a GUI of the music player. In that there is an option called “Open File”, By selecting an open file user can able to see local files and able to select a song that the user wants.

Pause/Resume Music:

After selecting the Mp3 file, It automatically plays the song. When the user selects a pause button, the song stops playing. After Resume, again the song continues to play.

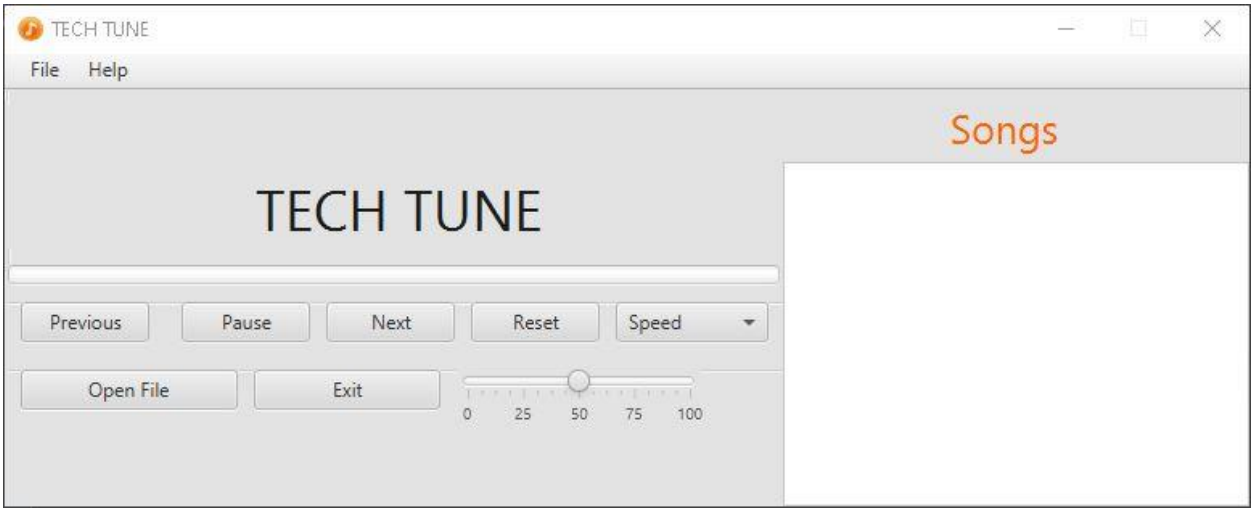
Previous /Next:

When the song playing, the user can change the song by selecting the buttons Previous and Next that will change the current running audio file to previous audio file or next audio file respectively to play.

- These are the basic functionalities that are any media player should consist of.

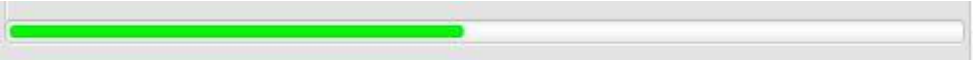
EXTERNAL INTERFACE REQUIREMENTS

User Interface

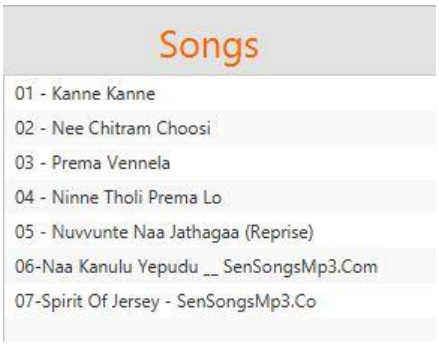


Initial UI: This is the view of the application whenever it is launched. No function will work for us until we open any mp3 file

Menu Bar: The menu bar will be displayed at the top of the audio player which consists of File, Help. In file we have reset and quit options and in help we have about option which tells about the creators of the audio player.

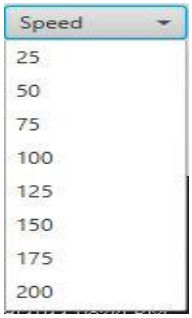


Progress Bar: We can also say this as Status Bar. It is used for displaying the duration of running audio file. User also seek in different position of file during the run time through this Progress Bar or Status Bar. The mouse click on any position on the status bar will be resulted as the running file will be seek to that time duration.



Playlist: After clicking the open file button you are allowed to select a mp3 file whenever you selects a mp3 audio file then the files which are present in the parent folder of that selected mp3 file will be added to queue or list as well as play list view as shown above.

Volume Slider: A slider which allows user to manipulate the volume of the current running audio file.



Speed Box: Because of this feature only user can able to change the speed of the running audio file.

Previous Button: Whenever user clicks previous button then the previous song to the present song in the playlist will be played.

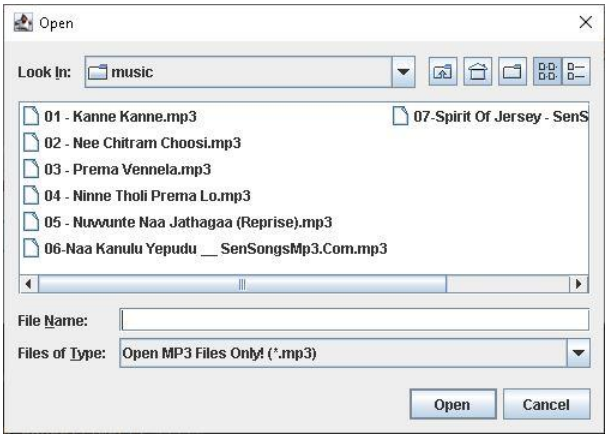
Pause/Resume Button: Pause button will be also acted as RESUME button. The text will be changed in the button from pause to resume whenever user click pause button to pause the song and vice versa.

Next Button: Whenever user clicks next button then the next song to the present song in the play list will be played.

Reset Button: Song will be played from the starting again.

Open file: open file is button which opens file chooser on clicking it.

Exit: closes the application.



File Chooser: This window will be popped up whenever user clicks on open file button and here we have to select an mp3 file only. And then the remaining audio files will be added to the list view automatically.

NON FUNCTIONAL INTERFACE

Performance

The music player requires a small amount of disk space for installation-30MB. It is characterized by fast loading and executing times. It is not a heavy program, and it can work while other programs are running.

Quickness

The system should be fast enough to play music and respond to any of the user's actions in any way without any buffering else it will be not a good experience.

Robustness

The system should be robust to deal with and act accordingly with common error scenarios like unavailable metadata, unsupported file types, etc.

Failure Handling

In case of failures, it is able to recover quickly with help of developers. For user purpose also we have included some error recovery method in the report.

Safety Requirements

The software should be able to restrict or warn if the user selects the next or previous button without opening a file. And it also restricts users to select files other than mp3 files.

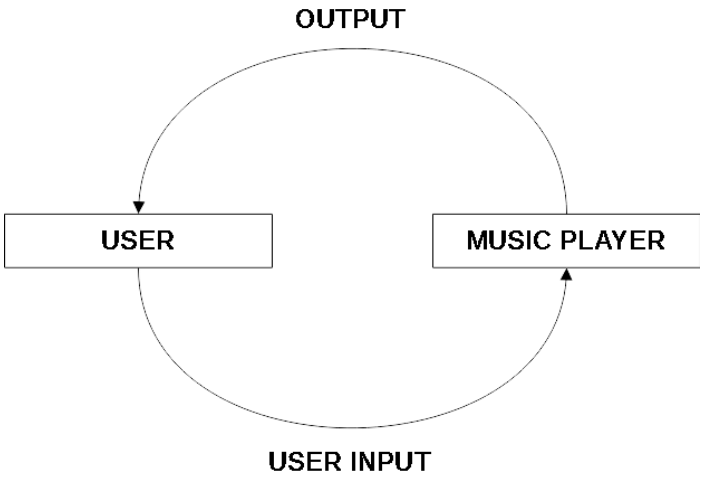
Software Quality Requirements

The music player is easy to use because it is a small program. Users can easily learn to use it. The software interface is quite friendly, simple, easily understood, and works fast. A music player is a fast and efficient program, and users can easily change its options to satisfy their needs some software quality attributes of a music player are:

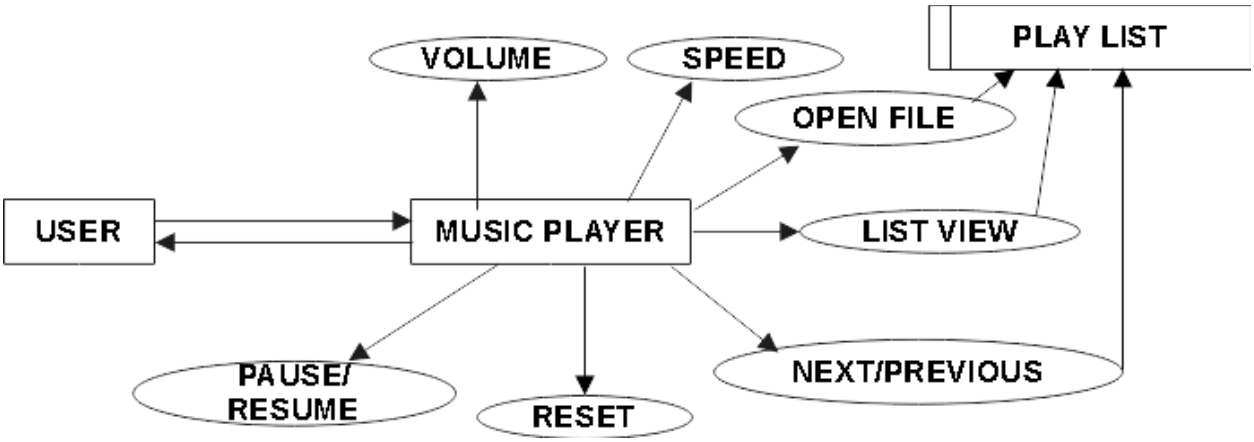
Memory Management: The system should not leak memory.
Compatibility: The system should peacefully co-exist with other software.

SYSTEM DESIGN

Data Flow Diagram

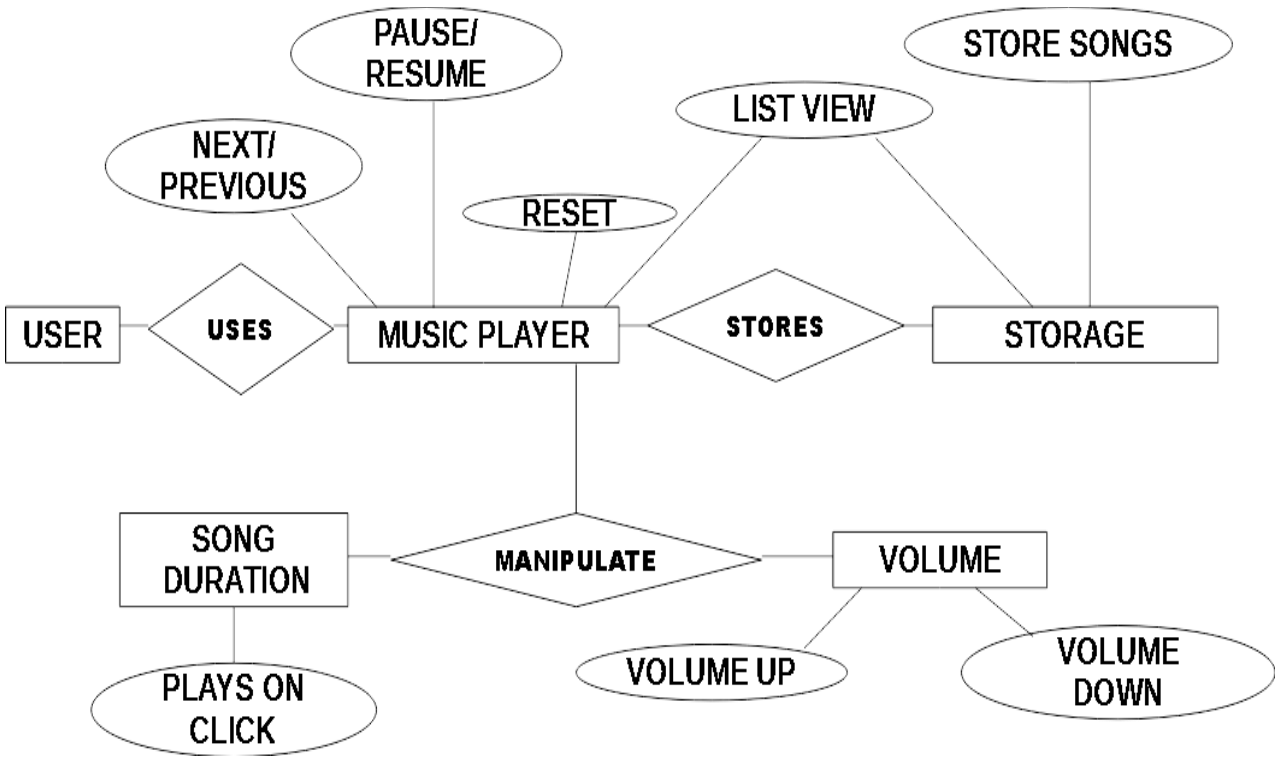


Level – 0 Diagram

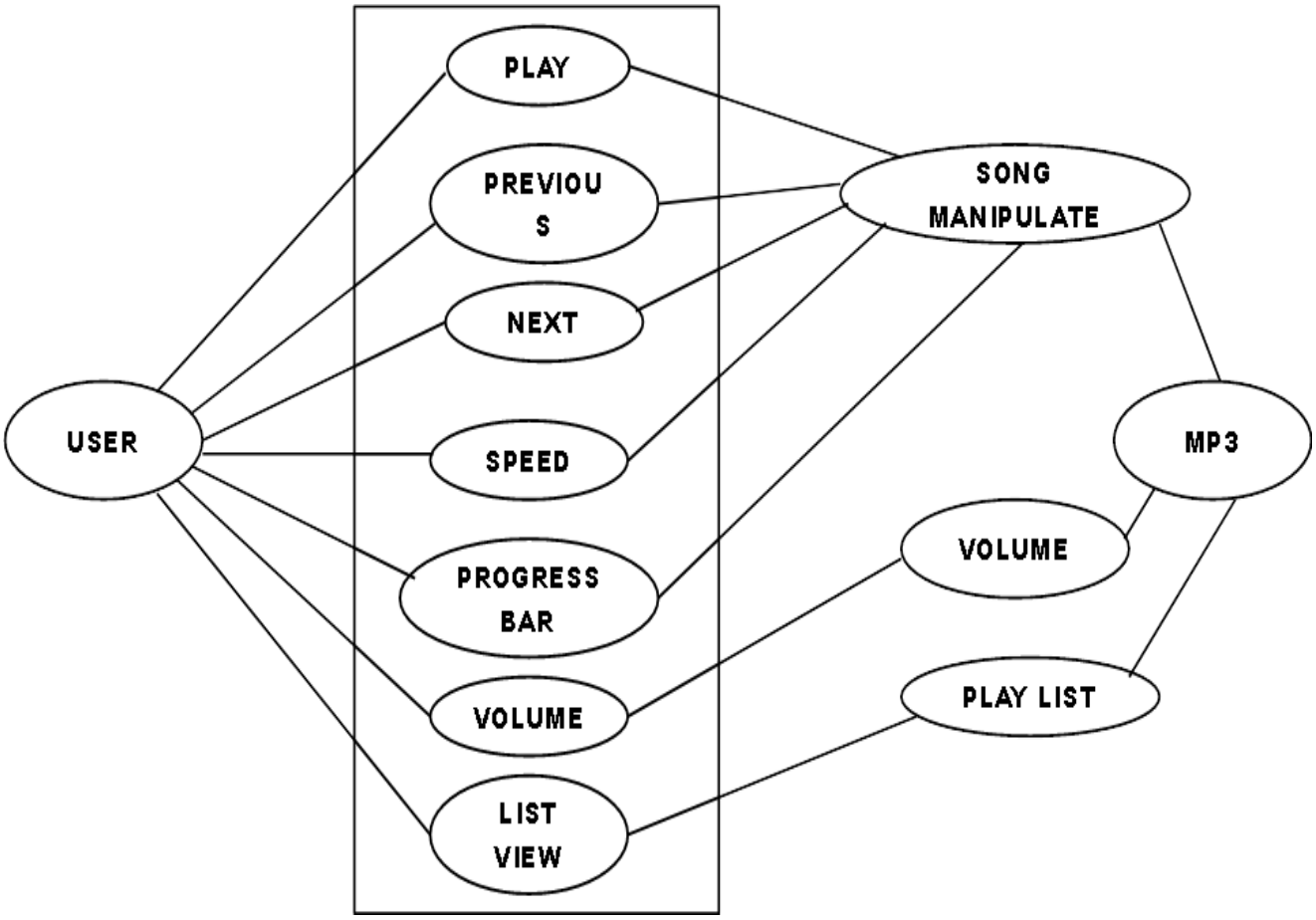


Level – 1 Diagram

ER Diagram


PAUSE/
RESUME
STORE SONGS
LIST VIEW
NEXT/
PREVIOUS
RESET
USER
USES
MUSIC PLAYER
STORES
STORAGE
SONG
DURATION
MANIPULATE
VOLUME
PLAYS ON
CLICK
VOLUME UP
VOLUME
DOWN

UML DIAGRAM



Use Case Diagram

Whole Project code is written in four files.

1. **Main.java**

   Our application start running from this file (main.java).
   Main file calls the controller file which consists of major functions like file opening, select songs, playing, timers and controllers.

2. **FileTypeFilter.java**

   Here files which are in parent folder of selected file will be sorted and those sorted files will be added to our list.
   It will sort out the files which are in format of .mp3(it means we only concentrate on mp3 files).

3. **Controller.java**

   This is the file where whole functioning will be done.
   It consists of major functions like open file, song play, timer, alerts, about, and controllers.
   Controllers resembles play, previous, pass, resume, speed box, and volume controller.

4. **Scene.xml**

   The User Interface(UI) is designed and coded in xml language.
   The UI consists of Buttons, Progress Bar, Speed Box, Volume Slider, List View, Menu Bar.
   We have used grid and split pane layouts to build user friendly UI.

MAIN.JAVA

```
package com.music;

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Scene;

import javafx.scene.image.Image;

import javafx.stage.Stage;

import java.io.IOException;


public class Main extends Application {

    @Override

    public void start(Stage stage) throws IOException {

        FXMLLoader fxmlLoader = new FXMLLoader(Main.class.getResource("Scene.fxml"));

        Scene scene = new Scene(fxmlLoader.load());

        stage.setTitle("TECH TUNE");

        stage.getIcons().add(new Image("music_icon.png"));

        stage.setResizable(false);

        stage.setScene(scene);

        stage.show();

    }

    public static void main(String[] args) {

        launch();

    }

}
```

FILETYPEFILTER.JAVA

```
package com.music;
import java.io.File;
import javax.swing.filechooser.FileFilter;
public class FileTypeFilter extends FileFilter{
     // File Extensions String
    private final String extension;
    // File Extension Description
    private final String description;
    // Constructor Method
    public FileTypeFilter(String extension, String description){
        // Set Constructor Values
        this.extension = extension;
        this.description = description;
    }
   @Override
    public boolean accept(File file) {
        // If File Is Returning Directory
        if(file.isDirectory()){
            return true;
        }
        // Return File Name with Extension
        return file.getName().endsWith(extension);
    }
    @Override
    public String getDescription() {
        // Return To Display File Type and Description
        return description + String.format(" (*%s)", extension);
    }
}
```

CONTROLLER.JAVA

```
package com.music;


import javafx.application.Platform;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.control.*;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.GridPane;
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;


import java.io.File;
import java.net.URL;
import java.util.*;
import java.util.Timer;


import javafx.util.Duration;


import javax.swing.*;


public class Controller implements Initializable {
    @FXML
    public Button previousBtn, playBtn, nextBtn, resetBtn, pauseBtn, openFile, exit;


    @FXML
    protected ComboBox<String> speedBox;


    @FXML
```

```
public Slider slider;

@FXML
public ProgressBar progressBar;

@FXML
public Label songNameLabel;

@FXML
public MenuItem openMenuItem,resetMenuItem, quitMenuItem, aboutMenuItem;

@FXML
public ListView<String> songsList;

@FXML
public GridPane gridPane;
int flag=0;
public  Media media;
public MediaPlayer mediaPlayer;
private File directory;
private File[] files;
private ArrayList<File> songs;
private int songNumber;
private final int[] speeds = {25, 50, 75, 100, 125, 150, 175, 200};

private Timer timer;
public Boolean running = false;

File songFile;
```

```
String currentDirectory = "V:\\intern\\project\\music";

@Override
public void initialize(URL location, ResourceBundle resources) {

    //  While no files are added,it will show the text-> FIVE STARS
    songNameLabel.setText("TECH TUNE");

    // An array list to store the audio files as objects.
    songs = new ArrayList<>();

    // For speeds. Audio File can be played at desired speeds.
    for (int speed : speeds) speedBox.getItems().add(Integer.toString(speed));

    // An action event to change the speed of audio file as user selects.
    speedBox.setOnAction(event -> speed());

    //  volume slider. To manipulate the volume of audio file.
    slider.valueProperty().addListener((observable, oldValue, newValue) -> {
        if(flag!=1){fileOpenAlert();return;}
        mediaPlayer.setVolume(slider.getValue() * 0.01);
    });

    //  progress bar. Show the run time of audio file
    progressBar.setStyle("-fx-accent : #00ff00");
    //progressBar.pop

    // On click event on progress bar to move forward or backward as user desired.
    progressBar.setOnMouseClicked(this::progressBarClick);//progressBar.setOnMouseClicked(mouseEvent ->
progressBarClick(mouseEvent));
```

```
    //  song list view. on mouse click event-> plays the selected song in the list view
    songsList.setOnMouseClicked(mouseEvent ->selectedSong());

}

public void openFile() {

    //  JFileChooser is a swig component, that opens the files to select.
    JFileChooser openFileChooser = new JFileChooser(currentDirectory);

    // FileTypeFilter is class created by developer to filter the mp3 audio files only.
    openFileChooser.setFileFilter(new FileTypeFilter(".mp3", "Open MP3 Files Only!"));
    int result = openFileChooser.showOpenDialog(null);
    // If condition will be true when the dialog box with files has popped up.
    if (result == JFileChooser.APPROVE_OPTION) {

        // SongFile will store the audio file which is selected by the user, when user opens folder or file.
        songFile = openFileChooser.getSelectedFile();

        // Check weather the selected file is mp3 or not if not it will simply return.
        if (!songFile.getName().endsWith(".mp3")){wrongFileAlert();return;}

        // check weather any songs present in arrayList or not.
        // And if present they will get cleared and new songs will be added
        // Actually works when user open files for the second time at single run.
        if(songs!=null){songs.clear();}
        // if any audio file is running while opening the files for two or more times then,
        // media player has to stop the current playing audio file and timer has to restart.
        if(running){cancelTimer(); mediaPlayer.stop();}
```

```
// song name label will be changed from previous to present playing audio file.
songNameLabel.setText(songFile.getName().substring(0,songFile.getName().length()-4));

// directory will store the path of the present playing audio file.
directory= new File(songFile.getParent());
// files store the all type of files in the directory or folder.
files=directory.listFiles();

if( files != null){

   //  flag variable, that defines weather any audio files is opened or not.
   flag=1;

   // while files are not null, in beginning we clear the list view of the songs.
   songsList.getItems().clear();
   for (File file : files) {
      // Check weather the file is mp3 or not.
      if (file.getName().contains(".mp3")) {
         // if the file is in the format of mp3 then it will be added to the songs arrayList.
         songs.add(file);
         // Simultaneously the name of the song will be added to the list view and displayed.
         songsList.getItems().add(file.getName().substring(0,file.getName().length()-4));

      }
      // Terminates when no file left unchecked.
   }

   // Searches for selected audio file in file chooser with in the songs arrayList
   for(int i=0; i<=songs.size()-1;i++){
      if(songFile.equals(songs.get(i))){
```

```
                // Whenever they got matched it will assign the song to media player and get ready to play.
                media = new Media(songFile.toURI().toString());
                mediaPlayer = new MediaPlayer(media);
                // Initially the volume of player is 50%
                mediaPlayer.setVolume(0.5);
                //Assigns the audio file number to songNumber.
                songNumber=i;
                // calling the function songPlay() to run or play the audio.
                songPlay();
            }
        }
    }
}

@FXML
public void songPlay() {

    // If there is no file in audio track, then it will simply return void.
    if(flag!=1){fileOpenAlert();return;}
    // if any song is already playing and calls for new song then we have to cancel the timer and restart it.
    if(running) {cancelTimer();}
    running = true;
    pauseBtn.setText("Pause");

    beginTimer();
    speed();
    // Adjusting the volume as per the value in the slider.
    mediaPlayer.setVolume(slider.getValue()*0.01);
    // playing the song which is already added in the media player.
```

```
    mediaPlayer.play();
}

private void progressBarClick(MouseEvent e) {

    // If there is no file in audio track, then it will simply return void.
    if(flag!=1)return;

    // Retrieving the parent of the progress bar for future reference
    Parent parent = progressBar.getParent();
    // if parent is not null then we will send that event to the parent.
    if (parent != null) parent.fireEvent(e);
    // x variable will store the x-axis value of progress bar, where click event happen.
    double x= e.getX();
    // width contains the total length of the progress bar.
    double width=progressBar.getWidth();
    double progression = (x/width);
    //ms contains the milliseconds of audio file where click event happens.
    int ms= (int) (progression*mediaPlayer.getTotalDuration().toMillis());
    // Duration is a class which stores the double value,
    // Here it stores milliseconds where click event happen by the mouse
    Duration duration= new Duration(ms);
    // playing the audio file, from where ever mouse click event happen by mouse
    mediaPlayer.seek(duration);
}

private void selectedSong() {

    // This function is called when user click on the song in the list view.
```

```
        // If there is no file in audio track, then it will simply return void.
        if(flag!=1)return;
        int temp=songNumber;
        // index of the selected song will be retrieved and stored in songNumber
        songNumber = songsList.getSelectionModel().getSelectedIndex();

        //Do nothing if user clicks on blank space in list view.
        if(songNumber>songs.size() || songNumber<0)return;
        // if user clicks on the audio file which is already playing then it will do nothing.
        // Means playing will continue without any disturb.
        if(temp==songNumber)return;

        // Have to stop the previously playing song firstly.
        mediaPlayer.stop();
        // Adding the selected song to media, with the help of songNumber. old media will be cleared.
        media = new Media(songs.get(songNumber).toURI().toString());
        // Adding the updated media to media player.
        mediaPlayer = new MediaPlayer(media);

        // Changing the song name label to present playing audio file or audio file which selected by user.
songNameLabel.setText(songs.get(songNumber).getName().substring(0,songs.get(songNumber).getName().length()
-4));
        // calling the song play function.
        songPlay();
    }

    @FXML
    public void previous() {

        // This method is called when user clicked on previous button.
```

```
        // Plays the preceding audio file to current playing audio file.

        // If there is no file in audio track, then it will simply return void.
        if(flag!=1){fileOpenAlert();return;}
        // i variable stores the pre audio file index for the present playing audio file.
        int i=songNumber==0 ? (songNumber=songs.size()-1):songNumber-- ;

        // stops the present playing audio file.
        mediaPlayer.stop();
        // if any song is already playing and calls for new song then we have to cancel the timer and restart it.
        if (running)cancelTimer();
        // Adding the selected song to media, with the help of songNumber. old media will be cleared.
        media = new Media(songs.get(songNumber).toURI().toString());
        // Adding the updated media to media player.
        mediaPlayer = new MediaPlayer(media);

        // Changing the song name label to present playing audio file or audio file which selected by user.

songNameLabel.setText(songs.get(songNumber).getName().substring(0,songs.get(songNumber).getName().length()
-4));
        // calling the song play function.
        songPlay();
    }

    @FXML
    public void next() {

        // This method is called when user clicked on next button.
        // If there is no file in audio track, then it will simply return void.
        if(flag!=1){fileOpenAlert();return;}
        // i variable stores the next audio file index for the present playing audio file.
```

```
        int i = (songs.size() - 1 > songNumber) ? (songNumber++) : (songNumber = 0);

        // stops the present playing audio file.
        mediaPlayer.stop();
        // if any song is already playing and calls for new song then we have to cancel the timer and restart it.
        if (running)cancelTimer();
        // Adding the selected song to media, with the help of songNumber. old media will be cleared.
        media = new Media(songs.get(songNumber).toURI().toString());
        // Adding the updated media to media player.
        mediaPlayer = new MediaPlayer(media);
        // Changing the song name label to present playing audio file or audio file which selected by user.
        Platform.runLater(() ->
songNameLabel.setText(songs.get(songNumber).getName().substring(0,songs.get(songNumber).getName().length()
-4)));
        // calling the song play function.
        songPlay();
    }

    public void pause() {

        // This method is called when user clicks on the pause button
        // when user clicks pause button then text will change from pause to resume and vice versa.

        // If there is no file in audio track, then it will simply return void.
        if(flag!=1){fileOpenAlert();return;}

        // if any audio file is already playing and calls for new song then we have to cancel the timer.
        if (running){
            cancelTimer();
            running = false;
            // Pause the media player until user click on the resume button.
```

```
            mediaPlayer.pause();
            // Change the text in pause button from pause from to resume.
            pauseBtn.setText("Resume");
        }
        // if no audio file is running else part will be executed.
        else {
            // Change the text in pause button form resume to pause
            pauseBtn.setText("Pause");
            // Calling the song play method.
            songPlay();
        }
    }

    public void reset() {

        // This method is executed when user clicks reset button,
        // It will reset the current playing audio file, to play it from beginning.

        // If there is no file in audio track, then it will simply return void.
        if(flag!=1){fileOpenAlert();return;}
        // Progress bar is reset to initial position.
        progressBar.setProgress(0);
        // Change the duration of song to zero seconds,
        // So that it will play form the beginning.
        mediaPlayer.seek(Duration.seconds(0));
    }

    public void speed() {

        // This method is executed when user clicks on the speed combo box.
```

```
    // It has a dropdown list, that are added before(25,50,100,125,150,175,200)

    // If there is no file in audio track, then it will simply return void.
    if(flag!=1){fileOpenAlert();return;}
    // if the value of speedBox is null then audio file will be played normally.
    if (speedBox.getValue() == null) {
        mediaPlayer.setRate(1);
    } else {
        mediaPlayer.setRate(Integer.parseInt(String.valueOf(speedBox.getValue())) * 0.01);
    }
}

private void beginTimer() {
    timer = new Timer();
    TimerTask timerTask = new TimerTask() {
        public void run() {
            running = true;
            try{
            double current = mediaPlayer.getCurrentTime().toSeconds();
            double end = mediaPlayer.getTotalDuration().toSeconds();
            progressBar.setProgress(current / end);
            if (current / end == 1) {
                // nextBtn.fire();
                cancelTimer();
                nextBtn.fire();
            }}catch(Exception x){close();}
        }
    };
    timer.scheduleAtFixedRate(timerTask, 0, 1000);
}
```

```
private void cancelTimer() {
    // changing the value of variable running to false.
    running = false;
    // cancelling the timer
    timer.cancel();
}

private void fileOpenAlert() {

    // Whenever user clicks on the buttons like next, previous, pause/resume, speeds, volume slider..so on,
    // without opening any file, it will result in pop up box which shows select file first.
    // Alert class is with the type of warning
    Alert fileOpenAlert = new Alert(Alert.AlertType.WARNING);
    fileOpenAlert.setTitle("Alert");
    fileOpenAlert.setHeaderText("No file selected!");
    fileOpenAlert.setContentText("Kindly Open a music file(mp3) first");
    fileOpenAlert.show();
}

private void wrongFileAlert(){

    //Whenever user selects other than mp3 file then it is not allowed to open that file.
    // Hence, it will open an alert that shows  "Wrong file selected!".
    Alert fileOpenAlert = new Alert(Alert.AlertType.WARNING);
    fileOpenAlert.setTitle("Alert");
    fileOpenAlert.setHeaderText("Wrong file selected!");
    fileOpenAlert.setContentText("You have to select only mp3 files");
    fileOpenAlert.show();
}
```

```
    @FXML
    private void aboutApp() {

        // This method is called when user, click about option which is present in menu bar.
        // Initializing  Alert class as information type.
        Alert aboutAlert = new Alert(Alert.AlertType.INFORMATION);
        aboutAlert.setTitle("About");
        aboutAlert.setHeaderText("Music Player");
        aboutAlert.setContentText("It is a simple music player. \nCreated by
\n\tR170340\n\tR170470\n\tR171004\n\tR171090");
        aboutAlert.show();
    }

    public void close() {
        Platform.exit();
        System.exit(0);
    }

}
```

Scene.fxml

```
<?xml version="1.0" encoding="UTF-8"?>


<?import javafx.scene.Cursor?>

<?import javafx.scene.control.Button?>

<?import javafx.scene.control.ComboBox?>

<?import javafx.scene.control.Label?>

<?import javafx.scene.control.ListView?>

<?import javafx.scene.control.Menu?>

<?import javafx.scene.control.MenuBar?>

<?import javafx.scene.control.MenuItem?>

<?import javafx.scene.control.ProgressBar?>

<?import javafx.scene.control.Separator?>

<?import javafx.scene.control.SeparatorMenuItem?>

<?import javafx.scene.control.Slider?>

<?import javafx.scene.control.SplitPane?>

<?import javafx.scene.effect.Blend?>

<?import javafx.scene.layout.ColumnConstraints?>

<?import javafx.scene.layout.GridPane?>

<?import javafx.scene.layout.HBox?>

<?import javafx.scene.layout.RowConstraints?>

<?import javafx.scene.layout.VBox?>

<?import javafx.scene.text.Font?>


<GridPane fx:id="gridPane" maxHeight="300.0" maxWidth="900.0" minHeight="281.0" minWidth="774.0"
prefHeight="281.0" prefWidth="774.0" snapToPixel="false" style="-fx-background-color: #e2e3e3;"
xmlns="http://javafx.com/javafx/17" xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.music.Controller">

  <columnConstraints>

    <ColumnConstraints hgrow="SOMETIMES" maxWidth="273.0" minWidth="0.0" prefWidth="6.0" />

    <ColumnConstraints hgrow="SOMETIMES" maxWidth="430.0" minWidth="10.0" prefWidth="373.0" />

    <ColumnConstraints hgrow="SOMETIMES" maxWidth="300.0" minWidth="10.0" prefWidth="129.0" />
```

```
    <ColumnConstraints hgrow="SOMETIMES" maxWidth="323.0" minWidth="10.0" prefWidth="243.0" />
    <ColumnConstraints hgrow="SOMETIMES" maxWidth="323.0" minWidth="10.0" prefWidth="63.0" />
  </columnConstraints>
  <rowConstraints>
    <RowConstraints maxHeight="121.0" minHeight="0.0" prefHeight="29.0" vgrow="SOMETIMES" />
    <RowConstraints maxHeight="319.0" minHeight="0.0" prefHeight="45.0" vgrow="SOMETIMES" />
    <RowConstraints maxHeight="319.0" minHeight="0.0" prefHeight="47.0" vgrow="SOMETIMES" />
    <RowConstraints maxHeight="307.0" minHeight="0.0" prefHeight="33.0" vgrow="SOMETIMES" />
    <RowConstraints maxHeight="327.0" minHeight="10.0" prefHeight="42.0" vgrow="SOMETIMES" />
    <RowConstraints maxHeight="327.0" minHeight="10.0" prefHeight="85.0" vgrow="SOMETIMES" />
  </rowConstraints>
  <children>
    <HBox maxHeight="1.7976931348623157E308" maxWidth="1.7976931348623157E308" prefHeight="63.0"
prefWidth="497.0" GridPane.columnIndex="1" GridPane.columnSpan="2" GridPane.rowIndex="5">
      <children>
        <Separator prefHeight="0.0" prefWidth="31.0" />
        <Button fx:id="openFile" mnemonicParsing="false" onAction="#openFile" prefHeight="25.0"
prefWidth="164.0" text="Open File">
          <effect>
            <Blend />
          </effect>
          <cursor>
            <Cursor fx:constant="HAND" />
          </cursor>
        </Button>
        <Separator prefHeight="2.0" prefWidth="21.0" />
        <Button fx:id="exit" mnemonicParsing="false" onAction="#close" prefHeight="25.0" prefWidth="145.0"
text="Exit">
          <cursor>
            <Cursor fx:constant="HAND" />
          </cursor>
        </Button>
```

```
        <Separator prefHeight="0.0" prefWidth="35.0" />
        <Slider fx:id="slider" prefHeight="38.0" prefWidth="181.0" showTickLabels="true"
showTickMarks="true" value="50.0">
          <cursor>
            <Cursor fx:constant="HAND" />
          </cursor>
        </Slider>
        <Separator prefHeight="1.0" prefWidth="79.0" />
      </children>
    </HBox>
    <VBox prefHeight="200.0" prefWidth="100.0" GridPane.columnSpan="2147483647">
      <children>
        <MenuBar maxHeight="1.7976931348623157E308" maxWidth="1.7976931348623157E308"
prefHeight="17.0" prefWidth="784.0">
          <menus>
            <Menu mnemonicParsing="false" text="File">
              <items>
                <MenuItem fx:id="openMenuItem" mnemonicParsing="false" text="Open" />
                <MenuItem fx:id="resetMenuItem" mnemonicParsing="false" text="Reset" />
                <SeparatorMenuItem mnemonicParsing="false" />
                <MenuItem fx:id="quitMenuItem" mnemonicParsing="false" onAction="#close" text="Quit" />
              </items>
            </Menu>
            <Menu mnemonicParsing="false" text="Help">
              <items>
                <MenuItem fx:id="aboutMenuItem" mnemonicParsing="false" onAction="#aboutApp"
text="About" />
              </items>
            </Menu>
          </menus>
        </MenuBar>
        <Separator orientation="VERTICAL" prefHeight="485.0" prefWidth="0.0" />
```

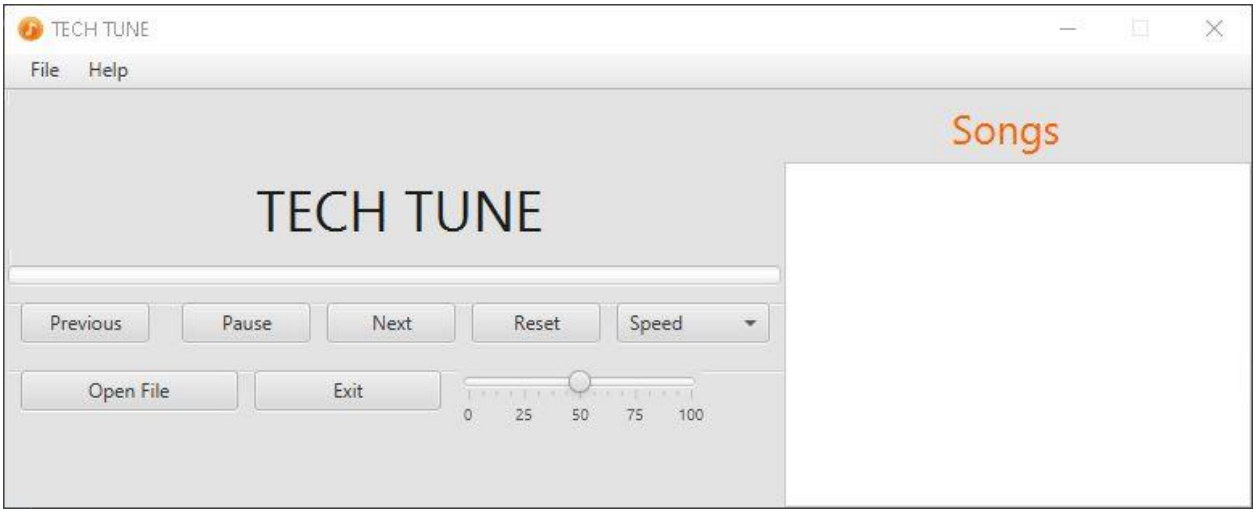
```
        </children>
      </VBox>
      <HBox fx:id="Hbox" prefHeight="100.0" prefWidth="525.0" GridPane.columnIndex="1"
GridPane.columnSpan="2" GridPane.rowIndex="2">
        <children>
          <SplitPane fx:id="split" prefHeight="51.0" prefWidth="6.0" style="-fx-background-color: #e2e3e3;" />
          <Label fx:id="songNameLabel" alignment="TOP_CENTER" contentDisplay="TOP"
maxHeight="1.7976931348623157E308" maxWidth="1.7976931348623157E308" prefHeight="36.0"
prefWidth="479.0" style="-fx-background-color: #e2e3e3;" text="Song Name" textFill="#1a1b1a">
            <font>
              <Font size="35.0" />
            </font>
          </Label>
        </children>
      </HBox>
      <VBox maxHeight="1.7976931348623157E308" maxWidth="1.7976931348623157E308" prefHeight="26.0"
prefWidth="484.0" style="-fx-background-color: #e2e3e3;" GridPane.columnIndex="1" GridPane.columnSpan="2"
GridPane.rowIndex="3">
        <children>
          <Separator orientation="VERTICAL" prefHeight="22.0" prefWidth="7.0" />
          <HBox prefHeight="44.0" prefWidth="503.0">
            <children>
              <SplitPane prefHeight="13.0" prefWidth="66.0" style="-fx-background-color: #e2e3e3;" />
              <ProgressBar fx:id="progressBar" prefHeight="13.0" prefWidth="582.0" progress="0.0">
                <cursor>
                  <Cursor fx:constant="HAND" />
                </cursor>
              </ProgressBar>
              <SplitPane prefHeight="13.0" prefWidth="66.0" style="-fx-background-color: #e2e3e3;" />
            </children>
          </HBox>
        </children>
```

```
    </VBox>
    <HBox maxHeight="1.7976931348623157E308" maxWidth="1.7976931348623157E308" prefHeight="54.0" prefWidth="627.0" GridPane.columnIndex="1" GridPane.columnSpan="2" GridPane.rowIndex="4">
      <children>
        <Separator prefHeight="3.0" prefWidth="16.0" />
        <Button fx:id="previousBtn" mnemonicParsing="false" onAction="#previous" prefWidth="200.0" text="Previous">
          <cursor>
            <Cursor fx:constant="HAND" />
          </cursor>
        </Button>
        <Separator prefHeight="1.0" prefWidth="37.0" />
        <Separator prefHeight="3.0" prefWidth="38.0" />
        <Button fx:id="pauseBtn" mnemonicParsing="false" onAction="#pause" prefWidth="200.0" text="Pause">
          <cursor>
            <Cursor fx:constant="HAND" />
          </cursor>
        </Button>
        <Separator prefHeight="3.0" prefWidth="47.0" />
        <Button fx:id="nextBtn" mnemonicParsing="false" onAction="#next" prefWidth="200.0" text="Next">
          <cursor>
            <Cursor fx:constant="HAND" />
          </cursor>
        </Button>
        <Separator prefHeight="3.0" prefWidth="42.0" />
        <Button fx:id="resetBtn" mnemonicParsing="false" onAction="#reset" prefHeight="25.0" prefWidth="200.0" text="Reset">
          <cursor>
            <Cursor fx:constant="HAND" />
          </cursor>
        </Button>
        <Separator prefHeight="3.0" prefWidth="52.0" />
```

```
        <ComboBox fx:id="speedBox" onAction="#speed" prefHeight="25.0" prefWidth="215.0"
promptText="Speed">

          <cursor>

            <Cursor fx:constant="HAND" />

          </cursor>

        </ComboBox>

        <Separator prefHeight="2.0" prefWidth="12.0" />

      </children>

    </HBox>

    <VBox GridPane.columnIndex="3" GridPane.columnSpan="2147483647" GridPane.rowIndex="1"
GridPane.rowSpan="2147483647">

      <children>

        <Label alignment="TOP_CENTER" contentDisplay="TOP" maxHeight="1.7976931348623157E308"
maxWidth="1.7976931348623157E308" minHeight="-Infinity" prefHeight="38.0" prefWidth="290.0" style="-fx-
background-color: #e2e3e3;" text="Songs  " textAlignment="CENTER" textFill="#ff6300">

          <font>

            <Font size="25.0" />

          </font>

        </Label>

        <ListView fx:id="songsList" maxHeight="1.7976931348623157E308"
maxWidth="1.7976931348623157E308" prefHeight="1000.0" prefWidth="1000.0" />

      </children>

    </VBox>

    <SplitPane prefHeight="13.0" prefWidth="66.0" style="-fx-background-color: #e2e3e3;"
GridPane.rowIndex="1" />

    <SplitPane prefHeight="13.0" prefWidth="66.0" style="-fx-background-color: #e2e3e3;"
GridPane.rowIndex="3" />

    <SplitPane prefHeight="13.0" prefWidth="66.0" style="-fx-background-color: #e2e3e3;"
GridPane.rowIndex="4" />

    <SplitPane prefHeight="13.0" prefWidth="66.0" style="-fx-background-color: #e2e3e3;"
GridPane.rowIndex="5" />

  </children>

</GridPane>
```

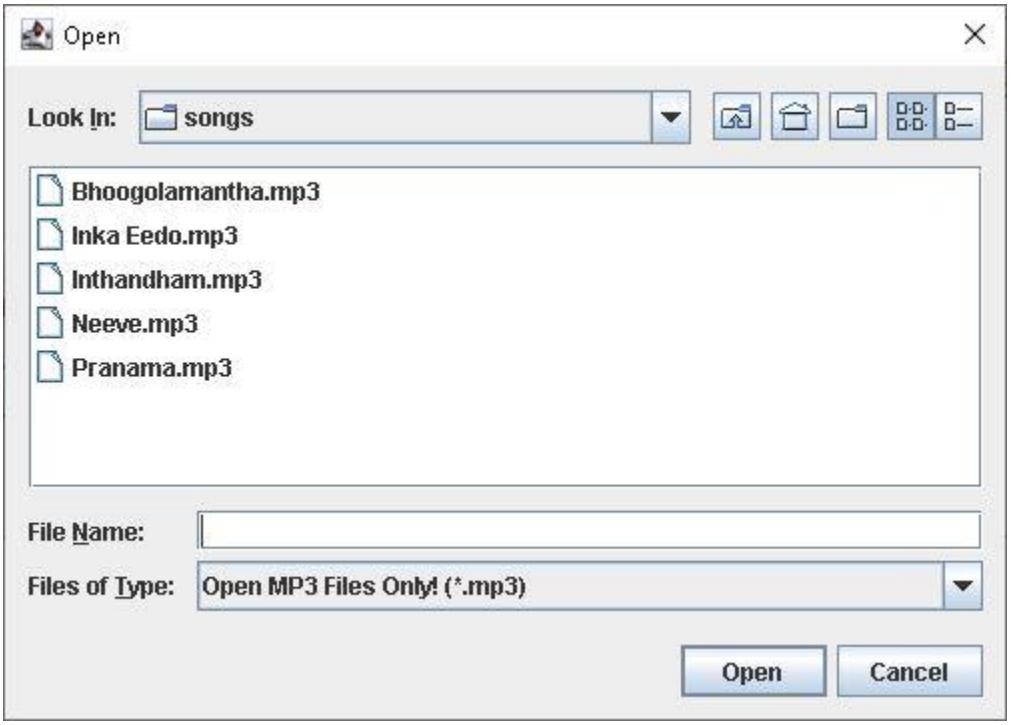
OUTPUT

INTIAL UI

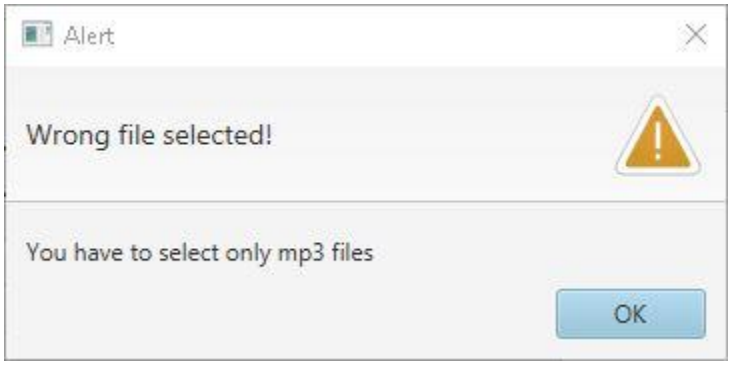


ALERT BOX

FILE CHOOSER



WRONG FILE ALERT



Conclusion:

TECH TUNE is a simple music player which is freeware and reliable to run audio files.

The user can easily selects the files from the local device with this software and can enjoy listening to the audio files. User can not able to select the file other than mp3 format files like mp4 wav zip etc.

It can automatically sort the audio files which are in the parent folder of the selected file. So that user doesn't need to open different file every time which are in the same folder.

Future Plans

- Accessing more audio format files.
- Playing video files.
- Adding keyboard input shortcuts to the player.
- Cutting the media files into small sub media files.
- Supports for capturing media device.