

## **Part B Micro-Project Report**

### **Music Player**

#### **1.0 Rationale:**

The introductory methodology for the music player project aims to develop a user-friendly and seamless solution for music management. It begins with extensive research to understand user preferences and requirements. By adopting an agile methodology, the development process is broken down into small, adaptable steps to remain flexible and responsive to feedback. The primary focus is on crafting an intuitive user interface and implementing real-time data handling capabilities to enhance usability. Development efforts encompass both frontend and backend systems, with features gradually expanded based on user feedback to accommodate evolving needs. Rigorous testing procedures are employed to identify and resolve any issues, while continuous integration of user feedback drives ongoing improvements. Comprehensive documentation accompanies the application to facilitate user guidance and ensure smooth operation. Through this approach, the music player project aims to deliver a comprehensive and user-friendly tool for managing and enjoying music collections, catering to the diverse preferences of users.

#### **2.0 Course Outcome Addressed:**

- Enhanced Product Visualization
- User-friendly Interface
- Accessibility on Android Devices
- UI Design Mastery
- Real-time Handling
- Problem Solving

### **3.0 Literature Review:**

1. **Android App Development:** Various resources provide insights into Android app development, including activity design, user interface development, and data handling techniques specific to music player applications.
2. **Media Playback:** Literature on media playback in Android apps offers guidance on implementing robust media controls, audio processing, and compatibility with different audio formats to ensure seamless playback performance.
3. **User Interface Design:** Research in user interface design explores principles and best practices for creating intuitive and visually appealing interfaces, focusing on elements such as playback controls, playlist management, and audio visualization.
4. **Audio Engineering:** Literature in audio engineering covers topics such as audio signal processing, equalization techniques, and sound synthesis, offering valuable knowledge for enhancing audio quality and implementing advanced audio features in music player apps.
5. **Human-Computer Interaction (HCI):** Studies in HCI provide insights into user interaction patterns, user preferences, and usability testing methodologies, guiding the design of user-friendly interfaces and features in music player applications.
6. **Mobile Computing:** Research in mobile computing discusses challenges and solutions related to resource management, performance optimization, and battery efficiency in mobile applications, offering strategies for enhancing the performance and efficiency of music player apps on Android devices.
7. **User Feedback Integration:** Literature on user feedback integration explores methods for collecting, analyzing, and prioritizing user feedback to drive continuous improvement and iteration in software development, informing the development process of music player apps based on user needs and preferences.

## 4.0 Actual Methodology Used:

- **activity\_main.xml:**

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <androidx.viewpager.widget.ViewPager
            android:id="@+id/view_pager"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:contentDescription="@string/view_pager_title"
            app:layout_behavior="@string/appbar_scrolling_view_behavior" />

        <include layout="@layout/layout_player_view" />

        <com.google.android.material.tabs.TabLayout
            android:id="@+id/tabs"
            style="?attr/tabStyle"
            android:layout_width="match_parent"
            android:layout_height="45dp"
            android:layout_weight="0"
            android:background="@color/colorBackgroundHigh"
            android:contentDescription="@string/tab_layout_title"
            app:tabIconTint="@color/tab_color"
            app:tabIndicator="@drawable/tab_indicator"
            app:tabIndicatorAnimationMode="linear"
            app:tabIndicatorFullWidth="true"
```

```
app:tabRippleColor="?colorPrimary" />
```

```
</LinearLayout>
```

```
</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

- **Mainactivity.java:**

```
package com.musicplayer;
```

```
import android.app.Activity;
import android.content.pm.PackageManager;
import android.os.Bundle;
import android.os.CountDownTimer;
import android.os.Handler;
import android.view.View;
import android.widget.ImageButton;
import android.widget.ImageView;
import android.widget.RelativeLayout;
import android.widget.TextView;
```

```
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.app.AppCompatActivity;
```

```
import com.atul.musicplayer.activities.PlayerDialog;
import com.atul.musicplayer.activities.QueueDialog;
import com.atul.musicplayer.adapter.MainPagerAdapter;
import com.atul.musicplayer.dialogs.SleepTimerDialog;
import com.atul.musicplayer.dialogs.SleepTimerDisplayDialog;
import com.atul.musicplayer.helper.MusicLibraryHelper;
import com.atul.musicplayer.helper.PermissionHelper;
import com.atul.musicplayer.helper.ThemeHelper;
import com.atul.musicplayer.listener.MusicSelectListener;
import com.atul.musicplayer.listener.PlayerDialogListener;
import com.atul.musicplayer.listener.SleepTimerSetListener;
import com.atul.musicplayer.model.Music;
import com.atul.musicplayer.player.PlayerBuilder;
import com.atul.musicplayer.player.PlayerListener;
import com.atul.musicplayer.player.PlayerManager;
import com.atul.musicplayer.viewmodel.MainViewModel;
import com.bumptech.glide.Glide;
import com.google.android.material.card.MaterialCardView;
import com.google.android.material.dialog.MaterialAlertDialogBuilder;
import com.google.android.material.progressindicator.LinearProgressIndicator;
import com.google.android.material.tabs.TabLayout;
```

```

import java.util.Collections;
import java.util.List;
import java.util.Locale;

public class MainActivity extends AppCompatActivity
    implements MusicSelectListener, PlayerListener, View.OnClickListener,
        SleepTimerSetListener, PlayerDialogListener {

    public static boolean isSleepTimerRunning;
    public static MutableLiveData<Long> sleepTimerTick;
    private static CountDownTimer sleepTimer;
    private RelativeLayout playerView;
    private ImageView albumArt;
    private TextView songName;
    private TextView songDetails;
    private ImageButton play_pause;
    private LinearProgressIndicator progressIndicator;
    private PlayerDialog playerDialog;
    private QueueDialog queueDialog;
    private PlayerBuilder playerBuilder;
    private PlayerManager playerManager;
    private boolean albumState;
    private MainViewModel viewModel;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTheme(ThemeHelper.getTheme(MPPreferences.getTheme(MainActivity.this)));

        AppCompatActivity.setDefaultNightMode(MPPreferences.getThemeMode(MainActivity.this))
        ;
        setContentView(R.layout.activity_main);
        MPConstants.musicSelectListener = this;

        viewModel = new ViewModelProvider(this).get(MainViewModel.class);

        if (PermissionHelper.hasReadStoragePermission(MainActivity.this)) {
            fetchMusicList();
            setUpUiElements();
        } else {
            manageStoragePermission(MainActivity.this);
        }

        if (!PermissionHelper.hasNotificationPermission(MainActivity.this)) {
            PermissionHelper.requestNotificationPermission(MainActivity.this);
        }

        albumState = MPPreferences.getAlbumRequest(this);

        MaterialCardView playerLayout = findViewById(R.id.player_layout);
        albumArt = findViewById(R.id.albumArt);
    }

```

```

        progressIndicator = findViewById(R.id.song_progress);
        playerView = findViewById(R.id.player_view);
        songName = findViewById(R.id.song_title);
        songDetails = findViewById(R.id.song_details);
        play_pause = findViewById(R.id.control_play_pause);
        ImageButton queue = findViewById(R.id.control_queue);

        play_pause.setOnClickListener(this);
        playerLayout.setOnClickListener(this);
        queue.setOnClickListener(this);
    }

    private void setPlayerView() {
        if (playerManager != null && playerManager.isPlaying()) {
            playerView.setVisibility(View.VISIBLE);
            onMusicSet(playerManager.getCurrentMusic());
        }
    }

    private void fetchMusicList() {
        new Handler().post() -> {
            List<Music> musicList = MusicLibraryHelper.fetchMusicLibrary(MainActivity.this);
            viewModel.setSongsList(musicList);
            viewModel.parseFolderList(musicList);
        });
    }

    public void setUpUiElements() {
        playerBuilder = new PlayerBuilder(MainActivity.this, this);
        MainPagerAdapter sectionsPagerAdapter = new MainPagerAdapter(
            getSupportFragmentManager(), this);

        ViewPager viewPager = findViewById(R.id.view_pager);
        viewPager.setAdapter(sectionsPagerAdapter);
        viewPager.setOffscreenPageLimit(3);

        TabLayout tabs = findViewById(R.id.tabs);
        tabs.setupWithViewPager(viewPager);

        for (int i = 0; i < tabs.getTabCount(); i++) {
            TabLayout.Tab tab = tabs.getTabAt(i);
            if (tab != null) {
                tab.setIcon(MPConstants.TAB_ICONS[i]);
            }
        }
    }

    public void manageStoragePermission(Activity context) {
        // required a dialog?
        if (PermissionHelper.requirePermissionRationale(context)) {
            new MaterialAlertDialogBuilder(context)

```

```

        .setTitle("Requesting permission")
        .setMessage("Enable storage permission for accessing the media files.")
        .setPositiveButton("Accept", (dialog, which) ->
PermissionHelper.requestStoragePermission(context)).show();
    } else {
        PermissionHelper.requestStoragePermission(context);
    }
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
    @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
        ThemeHelper.applySettings(this);
    }
}

@Override
protected void onDestroy() {
    super.onDestroy();
    if (playerBuilder != null)
        playerBuilder.unbindService();

    if (playerDialog != null)
        playerDialog.dismiss();

    if (queueDialog != null)
        queueDialog.dismiss();
}

@Override
public void playQueue(List<Music> musicList, boolean shuffle) {
    if (shuffle) {
        Collections.shuffle(musicList);
    }

    if (!musicList.isEmpty()) {
        playerManager.setMusicList(musicList);
        setPlayerView();
    }
}

@Override
public void addToQueue(List<Music> musicList) {
    if (!musicList.isEmpty()) {
        if (playerManager != null && playerManager.isPlaying())
            playerManager.addMusicQueue(musicList);
        else if (playerManager != null)
            playerManager.setMusicList(musicList);
    }
}

```

```

        setPlayerView();
    }
}

@Override
public void refreshMediaLibrary() {
    fetchMusicList();
}

@Override
public void onPrepared() {
    playerManager = playerBuilder.getPlayerManager();
    setPlayerView();
}

@Override
public void onStateChanged(int state) {
    if (state == State.PLAYING)
        play_pause.setImageResource(R.drawable.ic_controls_pause);
    else
        play_pause.setImageResource(R.drawable.ic_controls_play);
}

@Override
public void onPositionChanged(int position) {
    progressIndicator.setProgress(position);
}

@Override
public void onMusicSet(Music music) {
    songName.setText(music.title);
    songDetails.setText(
        String.format(Locale.getDefault(), "%s • %s",
            music.artist, music.album));
    playerView.setVisibility(View.VISIBLE);

    if (albumState)
        Glide.with(getApplicationContext())
            .load(music.albumArt)
            .centerCrop()
            .into(albumArt);

    if (playerManager != null && playerManager.isPlaying())
        play_pause.setImageResource(R.drawable.ic_controls_pause);
    else
        play_pause.setImageResource(R.drawable.ic_controls_play);
}

@Override
public void onPlaybackCompleted() {

```



```

    }

    @Override
    public void onRelease() {
        playerView.setVisibility(View.GONE);
    }

    @Override
    public void onClick(View v) {
        int id = v.getId();

        if (id == R.id.control_play_pause)
            playerManager.playPause();

        else if (id == R.id.control_queue)
            setUpQueueDialogHeadless();

        else if (id == R.id.player_layout)
            setUpPlayerDialog();
    }

    private void setUpPlayerDialog() {
        playerDialog = new PlayerDialog(this, playerManager, this);
        playerDialog.show();
    }

    @Override
    public void setTimer(int minutes) {
        if (!isSleepTimerRunning) {
            isSleepTimerRunning = true;
            sleepTimer = new CountDownTimer(minutes * 60 * 1000L, 1000) {
                @Override
                public void onTick(long l) {
                    if (sleepTimerTick == null) sleepTimerTick = new MutableLiveData<>();
                    sleepTimerTick.postValue(l);
                }

                @Override
                public void onFinish() {
                    isSleepTimerRunning = false;
                    playerManager.pauseMediaPlayer();
                }
            }.start();
        }
    }

    @Override
    public void cancelTimer() {
        if (isSleepTimerRunning && sleepTimer != null) {
            isSleepTimerRunning = false;
            sleepTimer.cancel();
        }
    }

```

```

    }
}

@Override
public MutableLiveData<Long> getTick() {
    return sleepTimerTick;
}

@Override
public void queueOptionSelect() {
    setUpQueueDialog();
}

@Override
public void sleepTimerOptionSelect() {
    setUpSleepTimerDialog();
}

private void setUpQueueDialog() {
    queueDialog = new QueueDialog(MainActivity.this, playerManager.getPlayerQueue());
    queueDialog.setOnDismissListener(v -> {
        if(!this.isDestroyed()) {
            playerDialog.show();
        }
    });

    playerDialog.dismiss();
    queueDialog.show();
}

private void setUpQueueDialogHeadless() {
    queueDialog = new QueueDialog(MainActivity.this, playerManager.getPlayerQueue());
    queueDialog.show();
}

private void setUpSleepTimerDialog() {
    if (MainActivity.isSleepTimerRunning) {
        setUpSleepTimerDisplayDialog();
        return;
    }
    SleepTimerDialog sleepTimerDialog = new SleepTimerDialog(MainActivity.this, this);
    sleepTimerDialog.setOnDismissListener(v -> {
        if(!this.isDestroyed()) playerDialog.show();
    });

    playerDialog.dismiss();
    sleepTimerDialog.show();
}

private void setUpSleepTimerDisplayDialog() {

```

```

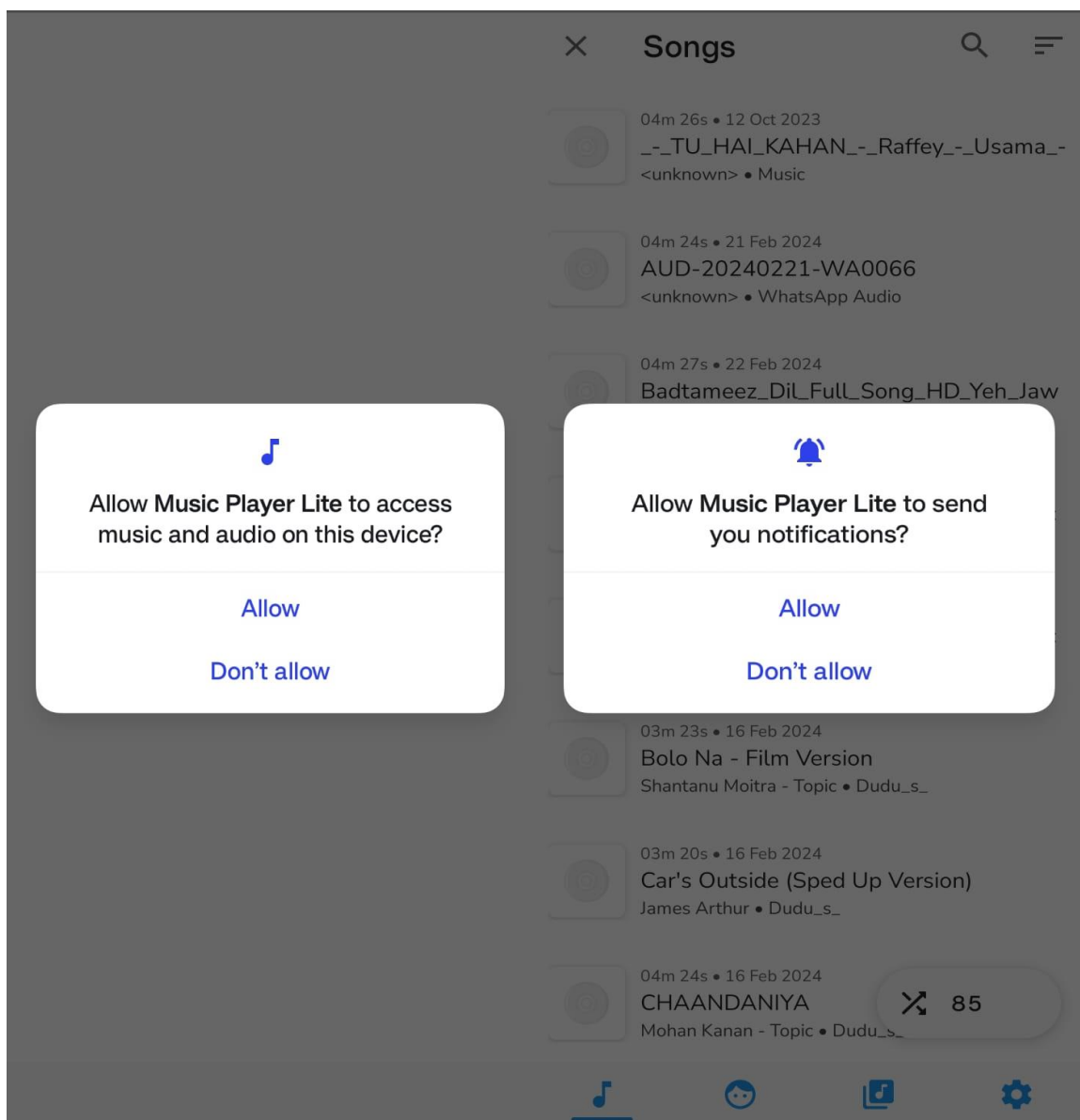
SleepTimerDisplayDialog sleepTimerDisplayDialog = new
SleepTimerDisplayDialog(MainActivity.this, this);
sleepTimerDisplayDialog.setOnDismissListener(v -> {
    if(!this.isDestroyed()) playerDialog.show();
});

playerDialog.dismiss();
sleepTimerDisplayDialog.show();
}
}

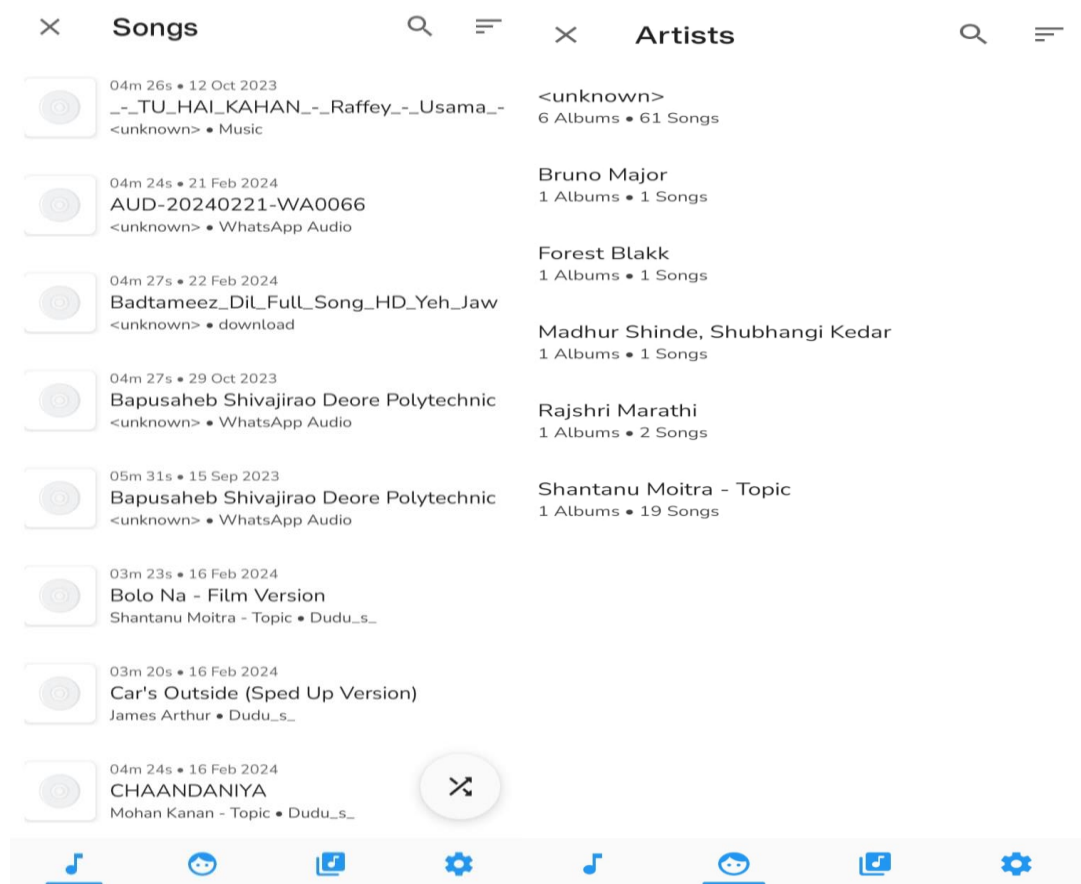
```

## 5.0 Output of Micro-Project:

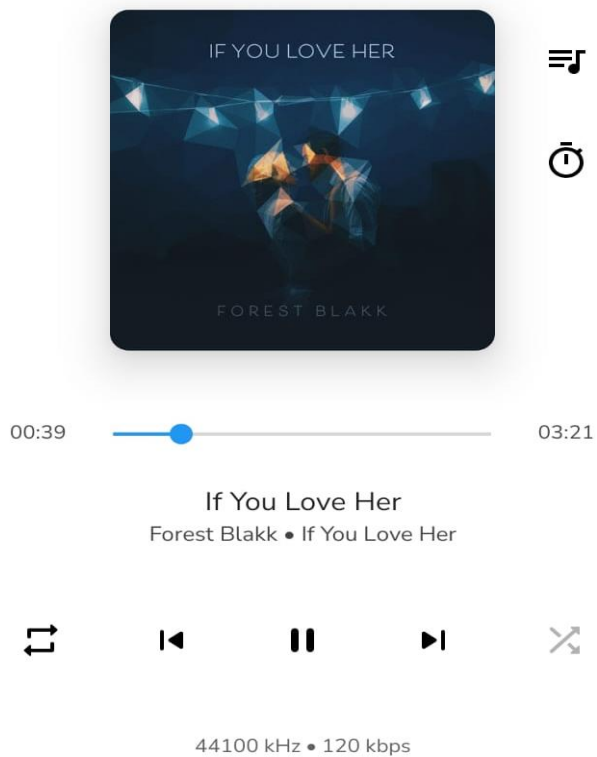
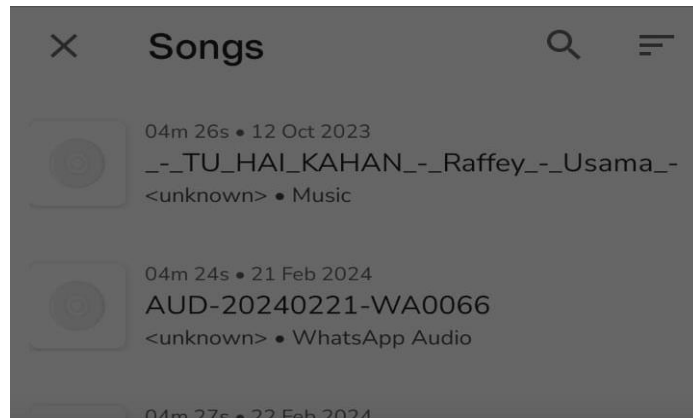
### 5.1 User permission for accessing local music, audio files and notifications












## 5.2 Local Audio, Music and Artists displayed in list view



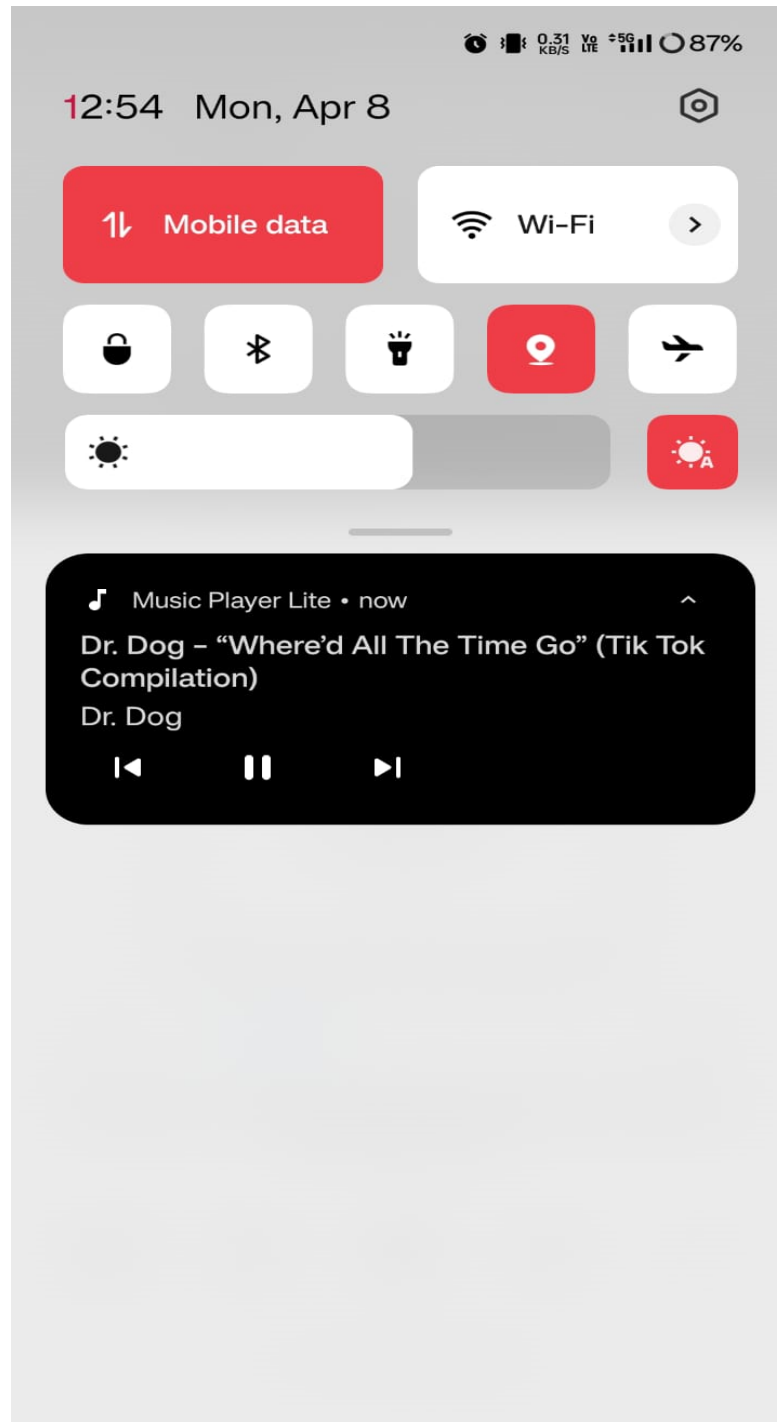
## 5.3 Music Player






## 5.4 Music queue, that will be played next


	If You Love Her Now playing	×
	Chipi_Chipi_Chapa_Chapa_but_fa <unknown>	×
	Lover Diljit Dosanjh	×
	TobyMac - Steal My Show (Lyrics) TobyMac	×
	Teri Hogaiyaan Vishal Mishra	×
	Steal The Show Lauv	×
	Chintamani Majha Madhur Shinde, Shubhangi Kedar	×
	Kitida Navyane Release - Topic	×
	Main Aur Tum Rono	×
	O Saajna - Lyrical   Broken But Zee Music Company	×
	Taylor Swift - cardigan Taylor Swift	×

## 5.4 Music controller in notification




## 5.5 Sleep timer, music will stop playing after a selected time period



Songs






04m 26s • 12 Oct 2023  
--TU\_HAI\_KAHAN--Raffey--Usama--  
<unknown> • Music




04m 24s • 21 Feb 2024  
AUD-20240221-WA0066  
<unknown> • WhatsApp Audio



04m 27s • 22 Feb 2024  
Badtameez\_DiL\_Full\_Song\_HD\_Yeh\_Jaw  
<unknown> • download



04m 27s • 29 Oct 2023  
Bapusaheb Shivajirao Deore Polytechnic  
<unknown> • WhatsApp Audio



05m 31s • 15 Sep 2023  
Bapusaheb Shivajirao Deore Polytechnic

SLEEP TIMER

Enter minutes to sleep in

5

10

15

20

ADD

## 6.0 Actual Resources Used:

Sr. No.	Name of Resource/Material	Specification	Quantity	Remark
1	Computer System	<b>OS:</b> Windows 11 (64 bit) <b>Processor:</b> Intel i5 11 <sup>th</sup> generation <b>RAM:</b> 8GB	1	-
2	Software	Android studio	1	-



## **7.0 Skilled developed/Learning out of this Microproject:**

1. **Android App Development:** Proficiency in developing Android applications, including activity design and user interface development.
2. **User Interface Design:** Skills in creating intuitive and visually appealing interfaces for mobile apps.
3. **Media Playback:** Implementation of robust media playback functionality for seamless music listening.
4. **Data Management:** Management of music metadata and playlists for efficient user interaction.
5. **Agile Methodology:** Working in small, adaptable steps to remain responsive to user feedback.
6. **Testing and Debugging:** Ability to test and debug Android apps for reliability.
7. **Continuous Improvement:** Integration of user feedback for iterative updates and enhancements.
8. **Documentation:** Creation of comprehensive documentation for user guidance.

## **8.0 Application of this Microproject:**

1. **Personal Music Management:** Users can utilize the music player app to manage their personal music collections, organizing tracks into playlists, and accessing them conveniently on their Android devices.
2. **Entertainment:** The music player app serves as a source of entertainment, allowing users to listen to their favorite songs, albums, and playlists anytime, anywhere, enhancing their overall music listening experience.
3. **Fitness and Workout Companion:** Users can use the music player app as a companion during workouts or fitness activities, creating energizing playlists to keep them motivated and focused.
4. **Commute and Travel Companion:** The app can accompany users during commutes or travels, providing a personalized soundtrack to their journeys and making the experience more enjoyable.

5. **Relaxation and Stress Relief:** Users can use the music player app to unwind and relax by listening to soothing music or ambient sounds, helping them alleviate stress and improve their mood.
6. **Social Gatherings and Parties:** The music player app can be used to set the mood and ambiance during social gatherings or parties, allowing users to create custom playlists tailored to the occasion.
7. **Learning and Productivity:** Users can use the app as a tool for learning or productivity, listening to educational podcasts, audio books, or instrumental music to enhance focus and concentration during study or work sessions.

## **9.0 Area of Future Improvement:**

1. **User Interface Refinement:** Continuously refine the interface for improved intuitiveness and visual appeal across devices.
2. **Advanced Audio Features:** Integrate support for high-resolution audio formats and audio effects to enhance sound quality.
3. **Cloud Integration:** Implement cloud storage for seamless access to music libraries across devices.
4. **Personalization:** Use machine learning for personalized music recommendations and curated playlists.
5. **Social Integration:** Enhance social sharing and collaboration features within the app.
6. **Accessibility:** Improve accessibility features for users with disabilities or diverse needs.
7. **Performance Optimization:** Optimize app performance for faster loading times and smoother playback.
8. **Feedback Loop:** Establish a feedback mechanism to prioritize user input for future updates.