Experiment 8: Multimedia- Audio Player

1. Objective

Develop an application that can play audio files stored on the device. This will cover aspects of multimedia handling in Android.

2. Steps to Complete the Experiment

1. Prepare Audio Files:

   Add audio files to your project's res/raw folder. If the folder doesn't exist, create it within res. Use compatible formats like MP3 or WAV.

2. Design the UI:

   Create a layout that includes Button elements for Play, Pause, and Stop functionalities.

   Optionally, include a SeekBar for audio progress and a TextView to display the current playback time and total duration.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
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"
    android:orientation="vertical"
```

```
android:padding="16dp"
tools:context=".MainActivity">

<Button
    android:id="@+id/buttonPlay"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Play" />

<Button
    android:id="@+id/buttonPause"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Pause" />

<Button
    android:id="@+id/buttonStop"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Stop" />

<!-- Optional SeekBar for playback progress -->
<SeekBar
    android:id="@+id/seekBarAudio"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

```
        android:layout_marginTop="16dp" />


</LinearLayout>
```

3. Initialize Media Player:

   In MainActivity.java, create a MediaPlayer instance. Use MediaPlayer.create(this, R.raw.your_audio_file) to initialize it with an audio file from the raw folder.

4. Implement Playback Controls:

   Set OnClickListener for each control button (Play, Pause, Stop) and define the corresponding actions using the MediaPlayer methods like start(), pause(), and stop().

   For the Stop functionality, ensure to prepare the MediaPlayer again after stopping, as calling stop() transitions the MediaPlayer into the Stopped state, and it needs to be prepared before it can play again.

5. Implement SeekBar Functionality (Optional):

   If using a SeekBar, update its progress to match the audio playback and allow the user to seek to different positions.

   Use a Handler to update the SeekBar and playback time display in real-time.

6. Handle Audio Focus (Optional):

   Request and manage audio focus to ensure your app plays nicely with other apps that might be playing audio. This involves handling audio interruptions gracefully.

7. Release Resources:

   Override the onDestroy method to release the MediaPlayer resources when the activity is destroyed, using mediaPlayer.release().

```
package com.yourpackage.name; // Replace with your actual package name
```

```
import androidx.appcompat.app.AppCompatActivity;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.os.Handler;
import android.view.View;
import android.widget.Button;
import android.widget.SeekBar;

public class MainActivity extends AppCompatActivity {

    private MediaPlayer mediaPlayer;
    private Button playButton, pauseButton, stopButton;
    private SeekBar seekBarAudio;
    private Handler handler = new Handler();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        playButton = findViewById(R.id.buttonPlay);
        pauseButton = findViewById(R.id.buttonPause);
        stopButton = findViewById(R.id.buttonStop);
        seekBarAudio = findViewById(R.id.seekBarAudio);
```

```
        mediaPlayer = MediaPlayer.create(this,
R.raw.your_audio_file); // Replace 'your_audio_file' with your
actual audio file name

        playButton.setOnClickListener(v -> {
            mediaPlayer.start();
            updateSeekBar();
        });

        pauseButton.setOnClickListener(v ->
mediaPlayer.pause());

        stopButton.setOnClickListener(v -> {
            mediaPlayer.stop();
            mediaPlayer =
MediaPlayer.create(getApplicationContext(),
R.raw.your_audio_file); // Reinitialize MediaPlayer
        });

        seekBarAudio.setMax(mediaPlayer.getDuration());
        seekBarAudio.setOnSeekBarChangeListener(new
SeekBar.OnSeekBarChangeListener() {
            @Override
            public void onProgressChanged(SeekBar seekBar, int
progress, boolean fromUser) {
                if (fromUser) {
```

```
                    mediaPlayer.seekTo(progress);
                }
            }

            @Override
            public void onStartTrackingTouch(SeekBar seekBar) {
                // Implementation not needed for this example
            }

            @Override
            public void onStopTrackingTouch(SeekBar seekBar) {
                // Implementation not needed for this example
            }
        });
    }

    private void updateSeekBar() {

seekBarAudio.setProgress(mediaPlayer.getCurrentPosition());
        if (mediaPlayer.isPlaying()) {
            Runnable updater = this::updateSeekBar;
            handler.postDelayed(updater, 1000);
        }
    }

    @Override
```

```
    protected void onDestroy() {
        super.onDestroy();
        if (mediaPlayer != null) {
            if (mediaPlayer.isPlaying()) {
                mediaPlayer.stop();
            }
            mediaPlayer.release();
        }
    }
}
```

8. Testing:

   Test the audio player functionalities: play, pause, and stop, ensuring the audio plays smoothly and the controls work as expected. If implemented, test the SeekBar for accuracy in displaying progress and seeking.

3. Explanation

The layout uses a LinearLayout with a vertical orientation, making the buttons and the SeekBar stack vertically.

Play Button (buttonPlay): Initiates audio playback.

Pause Button (buttonPause): Pauses the currently playing audio.

Stop Button (buttonStop): Stops the audio playback. You'll need to reinitialize the MediaPlayer if you wish to play the audio again after stopping.

SeekBar (seekBarAudio): Optionally included to allow users to see and control the playback progress. The SeekBar progress will need to be updated programmatically to reflect the current position of the audio being played.

MediaPlayer Initialization: The MediaPlayer is initialized with an audio file placed in the res/raw directory.

Play Button: Starts the audio playback using mediaPlayer.start() and calls updateSeekBar() to start updating the SeekBar progress.

Pause Button: Pauses the audio playback.

Stop Button: Stops the playback and reinitializes the MediaPlayer for future playback since calling stop() puts the MediaPlayer in the Stopped state, and it needs to be prepared again before it can play.

SeekBar: Reflects the progress of the audio playback. The user can seek to a specific part of the audio by dragging the seek thumb. The setMax() method sets the maximum value of the SeekBar to match the audio duration.

updateSeekBar: This method updates the SeekBar position to match the current position of the audio playback. It uses a Handler to repeatedly call itself every second as long as the audio is playing, ensuring the SeekBar stays in sync with the audio.

Resource Cleanup: In the onDestroy() method, the MediaPlayer resources are released to avoid leaking resources.