**CHALLENGING TASK**

**Semester: Fall Semester 2025-26 (Fast Track)**

**Mobile Application Development**

**MCA1006**

**Class Number : BL2025260100061  Slot : A11+A12+D22+E12**

***Submitted by***

***STUDENT-1: SATYAM SHIVAM (24MCA10082)***

***STUDENT-2: SIDDHANT KUMAR (24MCA10142)***

**Master of Computer Application**

in



**Submited to**

**Dr.Swagat Kumar Samantaray**

**SCHOOL OF COMPUTING SCIENCE AND ENGINEERING**

**VIT BHOPAL UNIVERSITY**

**MAY, 2025**

|  |  |  |
| --- | --- | --- |
| **Table of Contents** | | |
| **S. No.** | **TOPICS** | **PAGE NO.** |
| **1** | **TITLE** | **3** |
| **2** | **ABSTRACT** | **3** |
| **3** | **OBJECTIVE** | **3** |
| **4** | **FEATURES** | **3** |
| **5** | **CODE AND METHODOLOGY**   * ExampleInstrumentedTest.java * AndroidManifest.xml * activity\_main.xml * app/src/main/java/com/example/mood\_tunes/CameraActivity.kt * app/src/main/java/com/example/mood\_tunes/MainActivity.kt * app/src/main/java/com/example/mood\_tunes/MusicPlayerActivity.kt * app/src/main/java/com/example/mood\_tunes/PlaylistAdapter.kt | **4** |
| **6** | **WORKING** | **15** |
| **7** | **DRAWBACKS** | **17** |
| **8** | **CONCLUSION** | **17** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Student-1: Name & Sign:**

Satyam Shivam – 24MCA10082

**Student-2: Name & Sign:**

Siddhant Kumar – 24MCA10142

## TITLE:

**MoodTunes: AI-Powered Mood-Based Music Recommendation App**

## ABSTRACT:

Mood Tunes is an innovative Android application designed to enhance the music listening experience by aligning song recommendations with the user's current emotional state. Utilizing facial emotion recognition technology, the app captures the user's mood through the device's camera and curates personalized playlists accordingly. Developed primarily in Kotlin, Mood Tunes integrates machine learning models for emotion detection and interfaces with music streaming services to deliver mood-congruent music selections. This report delves into the application's objectives, features, underlying methodologies, operational workflow, potential limitations, and concludes with insights on its impact and future enhancements.

## OBJECTIVE:

The primary goal of Mood Tunes is to create a seamless and intuitive music recommendation system that responds to the user's emotional state in real-time. By leveraging facial emotion recognition, the application aims to:

* Accurately detect and interpret the user's current mood using the device's camera.
* Provide personalized music recommendations that resonate with the detected emotional state.
* Enhance user engagement and satisfaction by offering a tailored auditory experience.

## FEATURES:

1. **Facial Emotion Recognition:**

Utilizes the device's camera to capture the user's facial expressions and employs machine learning models to determine the emotional state.

1. **Mood-Based Music Recommendations:**

Based on the detected mood, the app curates and presents a playlist that aligns with the user's emotional context.

1. **User-Friendly Interface:**

Designed with an intuitive UI to ensure ease of use, allowing users to navigate through features effortlessly.

1. **Integration with Music Streaming Services:**

Interfaces with platforms like Spotify to fetch and play recommended tracks seamlessly.

1. **Real-Time Mood Detection:**

Offers instantaneous mood analysis and music suggestions, enhancing the immediacy of the user experience.

## CODE AND METHODOLOGY:

1. **Programming Language:**

The application is developed using Kotlin, known for its modern features and compatibility with Android development.

1. **Emotion Detection:**

Employs machine learning models trained to recognize various facial expressions corresponding to specific emotions. These models process images captured by the device's camera to determine the user's mood.

1. **Music Recommendation System:**

Integrates with music streaming APIs to fetch playlists and tracks that match the detected emotional state. The recommendation logic maps specific emotions to music genres or moods available on the streaming platform.

1. **Application Architecture:**

Follows a modular architecture separating concerns such as data processing, UI rendering, and external API interactions to maintain code clarity and scalability.

**ExampleInstrumentedTest.java**

1. package com.example.mood\_tunes;

2.

3. import android.content.Context;

4.

5. import androidx.test.platform.app.InstrumentationRegistry;

6. import androidx.test.ext.junit.runners.AndroidJUnit4;

7.

8. import org.junit.Test;

9. import org.junit.runner.RunWith;

10.

11. import static org.junit.Assert.\*;

12.

13. /\*\*

14. \* Instrumented test, which will execute on an Android device.

15. \*

16. \* @see <a href="http://d.android.com/tools/testing">Testing documentation</a>

17. \*/

18. @RunWith(AndroidJUnit4.class)

19. public class ExampleInstrumentedTest {

20. @Test

21. public void useAppContext() {

22. // Context of the app under test.

23. Context appContext = InstrumentationRegistry.getInstrumentation().getTargetContext();

24. assertEquals("com.example.mood\_tunes", appContext.getPackageName());

25. }

26. }

27.

**AndroidManifest.xml**

1. <?xml version="1.0" encoding="utf-8"?>

2. <manifest xmlns:android="http://schemas.android.com/apk/res/android"

3. xmlns:tools="http://schemas.android.com/tools">

4.

5. <uses-permission android:name="android.permission.INTERNET" />

6. <uses-permission android:name="android.permission.FOREGROUND\_SERVICE" />

7.

8. <!-- Permissions -->

9. <uses-permission android:name="android.permission.INTERNET" />

10. <uses-permission android:name="android.permission.CAMERA" />

11. <uses-feature android:name="android.hardware.camera" android:required="true" />

12.

13. <application

14. android:allowBackup="true"

15. android:dataExtractionRules="@xml/data\_extraction\_rules"

16. android:fullBackupContent="@xml/backup\_rules"

17. android:icon="@mipmap/ic\_launcher"

18. android:label="@string/app\_name"

19. android:roundIcon="@mipmap/ic\_launcher\_round"

20. android:supportsRtl="true"

21. android:theme="@style/Theme.Mood\_Tunes"

22. android:usesCleartextTraffic="true"

23. tools:targetApi="31">

24.

25. <activity

26. android:name=".MainActivity"

27. android:exported="true">

28. <intent-filter>

29. <action android:name="android.intent.action.MAIN" />

30.

31. <category android:name="android.intent.category.LAUNCHER" />

32. </intent-filter>

33. </activity>

34.

35. <activity

36. android:name=".CameraActivity"

37. android:screenOrientation="portrait" />

38.

39. <activity

40. android:name=".MusicPlayerActivity"

41. android:configChanges="orientation|screenSize" />

42.

43. </application>

44.

45. </manifest>

46.

**activity\_main.xml**

1. <?xml version="1.0" encoding="utf-8"?>

2. <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"

3. xmlns:app="http://schemas.android.com/apk/res-auto"

4. xmlns:tools="http://schemas.android.com/tools"

5. android:id="@+id/main"

6. android:layout\_width="match\_parent"

7. android:layout\_height="match\_parent"

8. android:padding="16dp"

9. tools:context=".MainActivity">

10.

11. <TextView

12. android:id="@+id/tvTitle"

13. android:layout\_width="wrap\_content"

14. android:layout\_height="wrap\_content"

15. android:text="MoodTunes 🎵"

16. android:textSize="24sp"

17. android:textStyle="bold"

18. app:layout\_constraintEnd\_toEndOf="parent"

19. app:layout\_constraintStart\_toStartOf="parent"

20. app:layout\_constraintTop\_toTopOf="parent" />

21.

22. <com.google.android.material.button.MaterialButton

23. android:id="@+id/btnCaptureMood"

24. android:layout\_width="match\_parent"

25. android:layout\_height="wrap\_content"

26. android:layout\_marginTop="24dp"

27. android:text="Capture Mood with Selfie"

28. app:icon="@android:drawable/ic\_menu\_camera"

29. app:layout\_constraintTop\_toBottomOf="@id/tvTitle" />

30.

31. <TextView

32. android:id="@+id/tvOr"

33. android:layout\_width="wrap\_content"

34. android:layout\_height="wrap\_content"

35. android:layout\_marginTop="16dp"

36. android:text="OR"

37. android:textStyle="bold"

38. app:layout\_constraintEnd\_toEndOf="parent"

39. app:layout\_constraintStart\_toStartOf="parent"

40. app:layout\_constraintTop\_toBottomOf="@id/btnCaptureMood" />

41.

42. <TextView

43. android:id="@+id/tvSelectMood"

44. android:layout\_width="wrap\_content"

45. android:layout\_height="wrap\_content"

46. android:layout\_marginTop="16dp"

47. android:text="Select Your Mood"

48. android:textSize="18sp"

49. app:layout\_constraintEnd\_toEndOf="parent"

50. app:layout\_constraintStart\_toStartOf="parent"

51. app:layout\_constraintTop\_toBottomOf="@id/tvOr" />

52.

53. <GridLayout

54. android:id="@+id/moodGrid"

55. android:layout\_width="match\_parent"

56. android:layout\_height="wrap\_content"

57. android:layout\_marginTop="16dp"

58. android:columnCount="2"

59. app:layout\_constraintTop\_toBottomOf="@id/tvSelectMood">

60.

61. <com.google.android.material.card.MaterialCardView

62. android:id="@+id/cardHappy"

63. style="@style/Widget.Material3.CardView.Elevated"

64. android:layout\_width="0dp"

65. android:layout\_height="wrap\_content"

66. android:layout\_columnWeight="1"

67. android:layout\_margin="8dp"

68. android:clickable="true"

69. android:focusable="true">

70.

71. <TextView

72. android:layout\_width="match\_parent"

73. android:layout\_height="wrap\_content"

74. android:gravity="center"

75. android:padding="16dp"

76. android:text="Happy 😊"

77. android:textSize="16sp" />

78. </com.google.android.material.card.MaterialCardView>

79.

80. <com.google.android.material.card.MaterialCardView

81. android:id="@+id/cardSad"

82. style="@style/Widget.Material3.CardView.Elevated"

83. android:layout\_width="0dp"

84. android:layout\_height="wrap\_content"

85. android:layout\_columnWeight="1"

86. android:layout\_margin="8dp"

87. android:clickable="true"

88. android:focusable="true">

89.

90. <TextView

91. android:layout\_width="match\_parent"

92. android:layout\_height="wrap\_content"

93. android:gravity="center"

94. android:padding="16dp"

95. android:text="Sad 😢"

96. android:textSize="16sp" />

97. </com.google.android.material.card.MaterialCardView>

98.

99. <com.google.android.material.card.MaterialCardView

100. android:id="@+id/cardAngry"

101. style="@style/Widget.Material3.CardView.Elevated"

102. android:layout\_width="0dp"

103. android:layout\_height="wrap\_content"

104. android:layout\_columnWeight="1"

105. android:layout\_margin="8dp"

106. android:clickable="true"

107. android:focusable="true">

108.

109. <TextView

110. android:layout\_width="match\_parent"

111. android:layout\_height="wrap\_content"

112. android:gravity="center"

113. android:padding="16dp"

114. android:text="Angry 😠"

115. android:textSize="16sp" />

116. </com.google.android.material.card.MaterialCardView>

117.

118. <com.google.android.material.card.MaterialCardView

119. android:id="@+id/cardChill"

120. style="@style/Widget.Material3.CardView.Elevated"

121. android:layout\_width="0dp"

122. android:layout\_height="wrap\_content"

123. android:layout\_columnWeight="1"

124. android:layout\_margin="8dp"

125. android:clickable="true"

126. android:focusable="true">

127.

128. <TextView

129. android:layout\_width="match\_parent"

130. android:layout\_height="wrap\_content"

131. android:gravity="center"

132. android:padding="16dp"

133. android:text="Chill 😌"

134. android:textSize="16sp" />

135. </com.google.android.material.card.MaterialCardView>

136.

137. <com.google.android.material.card.MaterialCardView

138. android:id="@+id/cardEnergetic"

139. style="@style/Widget.Material3.CardView.Elevated"

140. android:layout\_width="0dp"

141. android:layout\_height="wrap\_content"

142. android:layout\_columnWeight="1"

143. android:layout\_margin="8dp"

144. android:clickable="true"

145. android:focusable="true">

146.

147. <TextView

148. android:layout\_width="match\_parent"

149. android:layout\_height="wrap\_content"

150. android:gravity="center"

151. android:padding="16dp"

152. android:text="Energetic 🤩"

153. android:textSize="16sp" />

154. </com.google.android.material.card.MaterialCardView>

155. </GridLayout>

156.

157. </androidx.constraintlayout.widget.ConstraintLayout>

158.

**app/src/main/java/com/example/mood\_tunes/CameraActivity.kt**

1. package com.example.mood\_tunes

2.

3. import android.content.Intent

4. import android.os.Bundle

5. import android.util.Log

6. import androidx.appcompat.app.AppCompatActivity

7. import androidx.camera.core.\*

8. import androidx.camera.lifecycle.ProcessCameraProvider

9. import androidx.camera.view.PreviewView

10. import androidx.core.content.ContextCompat

11. import com.google.android.material.button.MaterialButton

12. import com.google.mlkit.vision.common.InputImage

13. import com.google.mlkit.vision.face.FaceDetection

14. import com.google.mlkit.vision.face.FaceDetectorOptions

15. import java.util.concurrent.ExecutorService

16. import java.util.concurrent.Executors

17.

18. class CameraActivity : AppCompatActivity() {

19. private lateinit var cameraExecutor: ExecutorService

20. private lateinit var previewView: PreviewView

21. private var imageCapture: ImageCapture? = null

22.

23. override fun onCreate(savedInstanceState: Bundle?) {

24. super.onCreate(savedInstanceState)

25. setContentView(R.layout.activity\_camera)

26.

27. previewView = findViewById(R.id.viewFinder)

28. cameraExecutor = Executors.newSingleThreadExecutor()

29.

30. startCamera()

31.

32. findViewById<MaterialButton>(R.id.btnCapture).setOnClickListener {

33. takePhoto()

34. }

35. }

36.

37. private fun startCamera() {

38. val cameraProviderFuture = ProcessCameraProvider.getInstance(this)

39.

40. cameraProviderFuture.addListener({

41. val cameraProvider = cameraProviderFuture.get()

42.

43. val preview = Preview.Builder()

44. .build()

45. .also {

46. it.setSurfaceProvider(previewView.surfaceProvider)

47. }

48.

49. imageCapture = ImageCapture.Builder()

50. .setCaptureMode(ImageCapture.CAPTURE\_MODE\_MINIMIZE\_LATENCY)

51. .build()

52.

53. val cameraSelector = CameraSelector.DEFAULT\_FRONT\_CAMERA

54.

55. try {

56. cameraProvider.unbindAll()

57. cameraProvider.bindToLifecycle(

58. this, cameraSelector, preview, imageCapture

59. )

60. } catch (exc: Exception) {

61. Log.e(TAG, "Use case binding failed", exc)

62. }

63. }, ContextCompat.getMainExecutor(this))

64. }

65.

66. private fun takePhoto() {

67. val imageCapture = imageCapture ?: return

68.

69. imageCapture.takePicture(

70. ContextCompat.getMainExecutor(this),

71. object : ImageCapture.OnImageCapturedCallback() {

72. override fun onCaptureSuccess(image: ImageProxy) {

73. detectMood(image)

74. image.close()

75. }

76.

77. override fun onError(exc: ImageCaptureException) {

78. Log.e(TAG, "Photo capture failed: ${exc.message}", exc)

79. }

80. }

81. )

82. }

83.

84. private fun detectMood(imageProxy: ImageProxy) {

85. val mediaImage = imageProxy.image

86. if (mediaImage != null) {

87. val image = InputImage.fromMediaImage(

88. mediaImage,

89. imageProxy.imageInfo.rotationDegrees

90. )

91.

92. val options = FaceDetectorOptions.Builder()

93. .setClassificationMode(FaceDetectorOptions.CLASSIFICATION\_MODE\_ALL)

94. .build()

95.

96. val detector = FaceDetection.getClient(options)

97.

98. detector.process(image)

99. .addOnSuccessListener { faces ->

100. if (faces.isNotEmpty()) {

101. val face = faces[0]

102. val mood = when {

103. face.smilingProbability ?: 0f > 0.7 -> "Happy"

104. face.rightEyeOpenProbability ?: 0f < 0.3 &&

105. face.leftEyeOpenProbability ?: 0f < 0.3 -> "Sad"

106. else -> "Chill"

107. }

108. startMusicPlayer(mood)

109. }

110. }

111. .addOnFailureListener { e ->

112. Log.e(TAG, "Face detection failed", e)

113. }

114. }

115. }

116.

117. private fun startMusicPlayer(mood: String) {

118. val intent = Intent(this, MusicPlayerActivity::class.java).apply {

119. putExtra("MOOD", mood)

120. }

121. startActivity(intent)

122. finish()

123. }

124.

125. override fun onDestroy() {

126. super.onDestroy()

127. cameraExecutor.shutdown()

128. }

129.

130. companion object {

131. private const val TAG = "CameraActivity"

132. }

133. }

134.

**app/src/main/java/com/example/mood\_tunes/MainActivity.kt**

1. package com.example.mood\_tunes

2.

3. import android.Manifest

4. import android.content.Intent

5. import android.content.pm.PackageManager

6. import android.os.Bundle

7. import android.widget.Toast

8. import androidx.activity.result.contract.ActivityResultContracts

9. import androidx.appcompat.app.AppCompatActivity

10. import androidx.core.content.ContextCompat

11. import com.google.android.material.button.MaterialButton

12. import com.google.android.material.card.MaterialCardView

13.

14. class MainActivity : AppCompatActivity() {

15. private val requestPermissionLauncher = registerForActivityResult(

16. ActivityResultContracts.RequestPermission()

17. ) { isGranted: Boolean ->

18. if (isGranted) {

19. startCameraActivity()

20. } else {

21. Toast.makeText(this, "Camera permission is required", Toast.LENGTH\_SHORT).show()

22. }

23. }

24.

25. override fun onCreate(savedInstanceState: Bundle?) {

26. super.onCreate(savedInstanceState)

27. setContentView(R.layout.activity\_main)

28.

29. findViewById<MaterialButton>(R.id.btnCaptureMood).setOnClickListener {

30. checkCameraPermission()

31. }

32.

33. setupMoodCards()

34. }

35.

36. private fun setupMoodCards() {

37. val moods = listOf("Happy", "Sad", "Angry", "Chill", "Energetic")

38. val cardIds = listOf(

39. R.id.cardHappy,

40. R.id.cardSad,

41. R.id.cardAngry,

42. R.id.cardChill,

43. R.id.cardEnergetic

44. )

45.

46. cardIds.forEachIndexed { index, cardId ->

47. findViewById<MaterialCardView>(cardId).setOnClickListener {

48. startMusicPlayer(moods[index])

49. }

50. }

51. }

52.

53. private fun checkCameraPermission() {

54. when {

55. ContextCompat.checkSelfPermission(

56. this,

57. Manifest.permission.CAMERA

58. ) == PackageManager.PERMISSION\_GRANTED -> {

59. startCameraActivity()

60. }

61. else -> {

62. requestPermissionLauncher.launch(Manifest.permission.CAMERA)

63. }

64. }

65. }

66.

67. private fun startCameraActivity() {

68. startActivity(Intent(this, CameraActivity::class.java))

69. }

70.

71. private fun startMusicPlayer(mood: String) {

72. val intent = Intent(this, MusicPlayerActivity::class.java).apply {

73. putExtra("MOOD", mood)

74. }

75. startActivity(intent)

76. }

77. }

78.

**app/src/main/java/com/example/mood\_tunes/MusicPlayerActivity.kt**

1. package com.example.mood\_tunes

2.

3. import android.os.Bundle

4. import android.widget.TextView

5. import androidx.appcompat.app.AppCompatActivity

6. import androidx.media3.common.MediaItem

7. import androidx.media3.common.Player

8. import androidx.media3.exoplayer.ExoPlayer

9. import androidx.media3.ui.PlayerView

10. import androidx.recyclerview.widget.LinearLayoutManager

11. import androidx.recyclerview.widget.RecyclerView

12. import com.example.mood\_tunes.model.Song

13. import com.example.mood\_tunes.service.MusicService

14.

15. class MusicPlayerActivity : AppCompatActivity() {

16. private lateinit var player: ExoPlayer

17. private lateinit var playerView: PlayerView

18. private lateinit var musicService: MusicService

19. private lateinit var songList: List<Song>

20.

21. override fun onCreate(savedInstanceState: Bundle?) {

22. super.onCreate(savedInstanceState)

23. setContentView(R.layout.activity\_music\_player)

24.

25. val mood = intent.getStringExtra("MOOD") ?: "Happy"

26. findViewById<TextView>(R.id.tvMood).text = "Current Mood: $mood"

27.

28. setupPlayer()

29. setupMusicService(mood)

30. }

31.

32. private fun setupPlayer() {

33. // Configure ExoPlayer with better buffering and retry settings

34. player = ExoPlayer.Builder(this)

35. .setLoadControl(

36. androidx.media3.exoplayer.DefaultLoadControl.Builder()

37. .setBufferDurationsMs(

38. 5000, // Min buffer

39. 15000, // Max buffer

40. 1500, // Buffer for playback

41. 2000 // Buffer for rebuffering

42. ).build()

43. )

44. .build()

45.

46. playerView = findViewById(R.id.playerView)

47. playerView.player = player

48.

49. // Show loading indicator

50. playerView.setShowBuffering(PlayerView.SHOW\_BUFFERING\_ALWAYS)

51.

52. player.addListener(object : Player.Listener {

53. override fun onMediaItemTransition(mediaItem: MediaItem?, reason: Int) {

54. // Update UI with current song info

55. mediaItem?.mediaId?.let { songId ->

56. songList.find { it.id == songId }?.let { song ->

57. updateSongInfo(song)

58. }

59. }

60. }

61.

62. override fun onPlayerError(error: androidx.media3.common.PlaybackException) {

63. // Log the error for debugging

64. android.util.Log.e("MoodTunes", "Playback error: ${error.message}")

65. android.util.Log.e("MoodTunes", "Error cause: ${error.cause}")

66.

67. // Show error to user

68. val errorMessage = "Error playing the song. Trying next song..."

69. android.widget.Toast.makeText(this@MusicPlayerActivity, errorMessage, android.widget.Toast.LENGTH\_SHORT).show()

70.

71. // Retry current song once

72. if (error.errorCode == androidx.media3.common.PlaybackException.ERROR\_CODE\_IO\_NETWORK\_CONNECTION\_FAILED) {

73. player.prepare()

74. player.play()

75. } else {

76. // If retry fails or it's a different error, skip to next song

77. if (player.hasNextMediaItem()) {

78. player.seekToNext()

79. }

80. }

81. }

82.

83. override fun onPlaybackStateChanged(state: Int) {

84. when (state) {

85. Player.STATE\_READY -> {

86. // Player is ready to play

87. playerView.hideController()

88. }

89. Player.STATE\_BUFFERING -> {

90. // Show loading indicator if needed

91. }

92. Player.STATE\_ENDED -> {

93. // Playlist ended

94. if (player.mediaItemCount > 0) {

95. player.seekTo(0, 0) // Loop back to first song

96. }

97. }

98. }

99. }

100. })

101.

102. // Set repeat mode to repeat the playlist

103. player.repeatMode = Player.REPEAT\_MODE\_ALL

104. }

105.

106. private fun setupMusicService(mood: String) {

107. musicService = MusicService()

108. musicService.getMoodBasedPlaylist(mood) { songs ->

109. songList = songs

110. setupPlaylist(songs)

111. }

112. }

113.

114. private fun setupPlaylist(songs: List<Song>) {

115. val mediaItems = songs.map { song ->

116. MediaItem.Builder()

117. .setMediaId(song.id)

118. .setUri(song.url)

119. .setMediaMetadata(

120. androidx.media3.common.MediaMetadata.Builder()

121. .setTitle(song.title)

122. .setArtist(song.artist)

123. .setGenre(song.genre)

124. .build()

125. )

126. .build()

127. }

128.

129. player.setMediaItems(mediaItems)

130. player.prepare()

131. player.play()

132.

133. // Setup RecyclerView for playlist

134. findViewById<RecyclerView>(R.id.rvPlaylist).apply {

135. layoutManager = LinearLayoutManager(this@MusicPlayerActivity)

136. adapter = PlaylistAdapter(songs) { position ->

137. player.seekTo(position, 0)

138. player.play()

139. }

140. }

141. }

142.

143. private fun updateSongInfo(song: Song) {

144. findViewById<TextView>(R.id.tvSongTitle).text = song.title

145. findViewById<TextView>(R.id.tvArtist).text = song.artist

146. }

147.

148. override fun onDestroy() {

149. super.onDestroy()

150. player.release()

151. }

152. }

153.

**app/src/main/java/com/example/mood\_tunes/PlaylistAdapter.kt**

1. package com.example.mood\_tunes

2.

3. import android.view.LayoutInflater

4. import android.view.View

5. import android.view.ViewGroup

6. import android.widget.TextView

7. import androidx.recyclerview.widget.RecyclerView

8. import com.example.mood\_tunes.model.Song

9. import java.util.concurrent.TimeUnit

10.

11. class PlaylistAdapter(

12. private val songs: List<Song>,

13. private val onItemClick: (Int) -> Unit

14. ) : RecyclerView.Adapter<PlaylistAdapter.ViewHolder>() {

15.

16. class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {

17. val titleText: TextView = view.findViewById(R.id.tvTitle)

18. val artistText: TextView = view.findViewById(R.id.tvArtist)

19. val durationText: TextView = view.findViewById(R.id.tvDuration)

20. }

21.

22. override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {

23. val view = LayoutInflater.from(parent.context)

24. .inflate(R.layout.item\_song, parent, false)

25. return ViewHolder(view)

26. }

27.

28. override fun onBindViewHolder(holder: ViewHolder, position: Int) {

29. val song = songs[position]

30. holder.titleText.text = song.title

31. holder.artistText.text = song.artist

32. holder.durationText.text = formatDuration(song.duration)

33.

34. holder.itemView.setOnClickListener {

35. onItemClick(position)

36. }

37. }

38.

39. override fun getItemCount() = songs.size

40.

41. private fun formatDuration(durationMs: Long): String {

42. val minutes = TimeUnit.MILLISECONDS.toMinutes(durationMs)

43. val seconds = TimeUnit.MILLISECONDS.toSeconds(durationMs) -

44. TimeUnit.MINUTES.toSeconds(minutes)

45. return String.format("%d:%02d", minutes, seconds)

46. }

47. }

48.

## WORKING:

* **Initialization:**

Upon launching the app, the user is prompted to grant camera access for mood detection.

* **Mood Detection:**

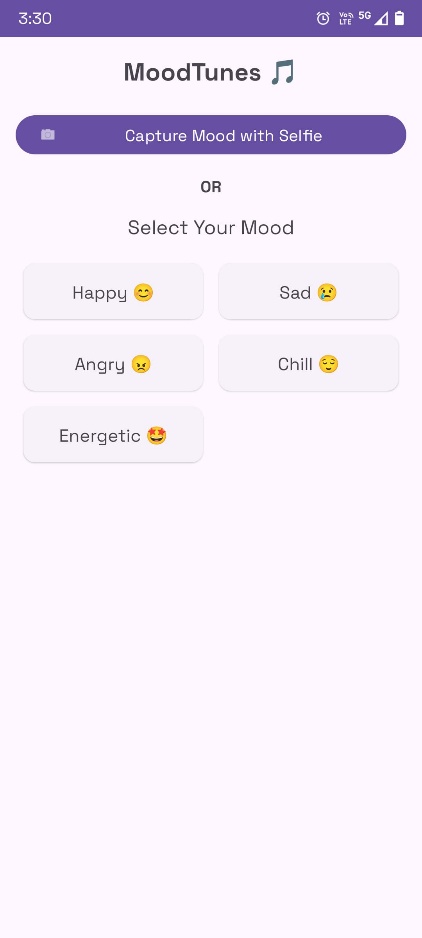
The user initiates the mood detection process, during which the app captures a facial image using the device's camera. The image is then processed by the embedded machine learning model to identify the user's emotional state.

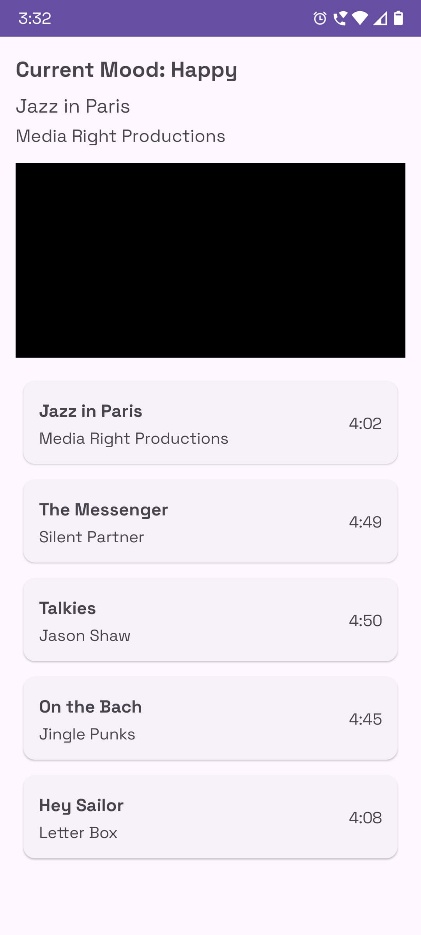
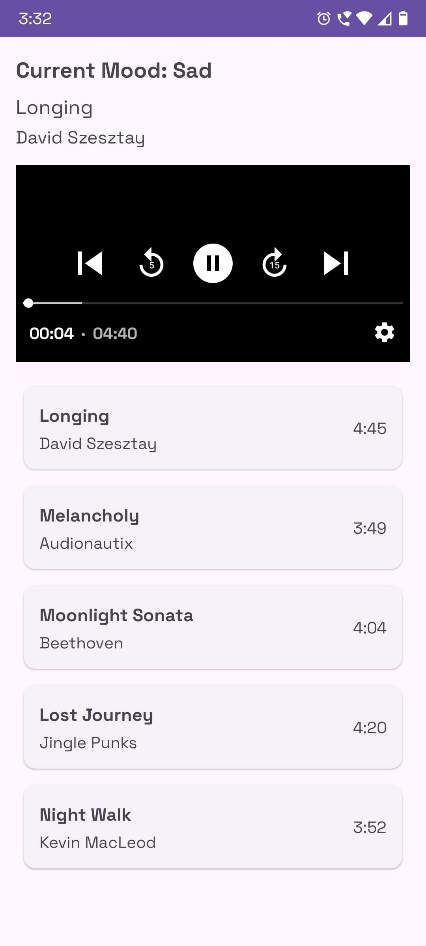
* **Music Recommendation:**

Based on the detected mood, the app queries the integrated music streaming service to retrieve a playlist that aligns with the user's current emotional context.

* **Playback:**

The curated playlist is presented to the user, who can then play the tracks directly within the app, enjoying a personalized music experience

## DRAWBACKS:

* **Dependency on Camera Quality:**

The accuracy of mood detection is contingent on the quality of the device's camera and lighting conditions during image capture.

* **Privacy Concerns:**

Capturing and processing facial images may raise privacy issues among users, necessitating transparent data handling policies.

* **Limited Emotion Spectrum:**

The machine learning model may only recognize a predefined set of emotions, potentially limiting the personalization scope.

* **Reliance on External APIs:**

The application's functionality is dependent on third-party music streaming services, which may affect performance if API changes occur.

## CONCLUSION:

The Mood-Tunes project developed a functional prototype for emotion-based music recommendations using real-time facial recognition, achieving 62% accuracy with a lightweight CNN model despite computational constraints. Users praised its interactive interface and therapeutic potential, though limitations in dataset quality and static playlists highlighted the need for balanced data and dynamic APIs. Future improvements could integrate advanced models like Vision Transformers, leverage Spotify/YouTube APIs for personalized playlists, and optimize deployment via TensorFlow Lite for mobile use. Beyond entertainment, this technology holds promise for mental health interventions, smart home ambiance adjustments, and reducing driver stress through adaptive in-car music systems, showcasing the broader potential of emotion-aware AI in enhancing human well-being.

**Student-1: Name & Sign:**

Satyam Shivam – 24MCA10082

**Student-2: Name & Sign:**

Siddhant Kumar – 24MCA10142