Tonelisten Android Framework Guide

Introduction:

Tonelisten is an android framework that can be used to integrate with Android applications. It was primarily written in Java for Android. The main concept behind Tonelisten is to display information to the user when it detects specific frequencies of tones being played around the user's android device microphone.

Pre-requisites:

- **Client key:** Obtain the client key from Tone Framework official Site.
- **Tonelisten Framework:** The Framework files can be obtained from the Tone Framework official Site.
- Minimum supported Android API level 22.

Embed Tonelisten Framework Android:

To begin, you need to add the Tonelisten framework into your project. The framework files can be obtained from the Tone Framework official Site.

To add the Tonelisten framework into your project's file structure, import the files to the project libs directory, under app directory.

Add Tonelisten Framework as dependency:

In Android Studio:

1. Create a new directory as "libs" inside the app directory of your project
2. Paste an AAR file (Tonelisten) inside the "libs" directory
3. Add the below line inside the project settings.gradle

```
dependencyResolutionManagement {

    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
```

```
    repositories {
        google()
        mavenCentral()
        flatDir {
            dirs("libs") //Fetch libraries inside libs directory
        }
    }
}
```

4. Add the below line inside the app's build.gradle

```
 //Kotlin Gradle
 implementation(files("libs/tonelisten-release.aar"))
 implementation("com.google.android.gms:play-services-location:$gms_location_version")
 implementation("androidx.room:room-runtime:$room_version")
 implementation("androidx.room:room-guava:$room_version")
 implementation("com.google.guava:guava:$guava_version")
 implementation("com.android.support:multidex:$multidex_version")
 implementation("com.google.code.gson:gson:$gson_version")
 implementation("com.squareup.okhttp3:okhttp:$okhttp_version")
 implementation("com.squareup.picasso:picasso:$picasso_version")
implementation("com.google.android.material:material:$material_version")

 //Groovy Gradle
 implementation files("libs/tonelisten-release.aar")
 implementation "com.google.android.gms:play-services-location:$gms_location_version"
 implementation "androidx.room:room-runtime:$room_version"
 implementation "androidx.room:room-guava:$room_version"
```

```
    implementation "com.google.guava:guava:$guava_version"

    implementation "com.android.support:multidex:$multidex_version"

    implementation "com.google.code.gson:gson:$gson_version"

    implementation "com.squareup.okhttp3:okhttp:$okhttp_version"

    implementation "com.squareup.picasso:picasso:$picasso_version"

    implementation "com.google.android.material:material:$material_version"
```

Add Device permissions:

- Permissions can be set as needed.
- Open the file app/src/main/AndroidManifest.xml, and add the following code:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.POST_NOTIFICATIONS" />
<uses-permission android:name="android.permission.CAPTURE_AUDIO_HOTWORD"
   tools:ignore="ProtectedPermissions" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<!--  Android 14, it must specify appropriate foreground service types -->
<uses-permission     android:name="android.permission.FOREGROUND_SERVICE_LOCATION" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE_MICROPHONE" />
```

```
   <application>
      <service
         android:name="com.strutbebetter.tonelisten.core.ToneServiceManager"
          android:enabled="true"
          android:exported="false"
          android:foregroundServiceType="location|microphone"
          android:process=":ToneListeningService"
          android:usesCleartextTraffic="true">
      </service>
   </application>
```

Note: For Android 6.0 or later, some important permissions must be requested at runtime rather than declared statically in the file AndroidMainfest.xml, therefore, you need to add the following code to do so (requestPermissions is a method of an Android Activity).

ToneFramework Components:

- **ToneFramework:** class serves as a central component for integrating and managing the ToneFramework within an Android application. The ToneFramework class encapsulates functionalities related to tone detection and handling within an Android application. It provides methods to start, restart, and finish the tone service, as well as to check permissions and handle tone responses. Additionally, it facilitates the setting of a Tone Tag Interface for receiving and processing tone tags.
- **ToneUIEventListener:** interface defines methods that allow classes to receive notifications when a tone is detected, when a tone tag is received, or when a bad code is detected. Classes implementing this interface can then respond to these events accordingly, such as displaying information related to the detected tone, processing tone tags, or handling bad codes.

Initialize Tone SDK:

- In the '**MainActivity'** class, the '**toneFramework'** object of type '**ToneFramework'** is initialized in the onCreate method.
- It's initialized with a reference to the '**MainActivity'** context.

```
public class MainActivity extends AppCompatActivity {
   private ToneFramework toneFramework;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       //initialize tone SDK
       toneFramework = new ToneFramework(MainActivity.this);
       //check permission
       toneFramework.checkPermission(ToneFramework.TONE_PERMISSION_CODE, this);
   }
}
```

- To set client ID, Pass Client ID as param to `setClientId()` function in the tone framework in `MainActivity`.

```
toneFramework.setClientId(clientID);
```

Check Permission:

- The '**checkPermission'** method of '**toneFramework'** is called to check for permissions required by the Tone SDK.
- It expects a permission code and the '**MainActivity'** context (**this**).
- This method requests access for Location & Microphone Access.
- ToneFramework won't work without this permission.

```
toneFramework.checkPermission(ToneFramework.TONE_PERMISSION_CODE, this);
```

- Request Code Check: The code verifies if the incoming request code matches `ToneFramework.TONE_PERMISSION_CODE`, ensuring it's handling the expected permission request.

```
@RequiresApi(api = Build.VERSION_CODES.O)
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
                                       @NonNull int[] grantResults) {
 super.onRequestPermissionsResult(requestCode, permissions, grantResults);

 if (requestCode == ToneFramework.TONE_PERMISSION_CODE) {
   // Checking whether the user granted the permission or not.
   if (grantResults.length > 0 && (grantResults[0] + grantResults[1] ==
PackageManager.PERMISSION_GRANTED)) {
     // Start Service
     toneFramework.start();
   } else {
     Toast.makeText(MainActivity.this, "Permission Denied", Toast.LENGTH_SHORT).show();
   }
 }
}
```

Start Service:

- Checks if the request code matches **ToneFramework.TONE_PERMISSION_CODE**.
- If permissions are granted, call `toneFramework.start()` to initiate the service.
- If permissions are denied, display a message to grant permission.

```
@RequiresApi(api = Build.VERSION_CODES.O)
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
                                       @NonNull int[] grantResults) {
 super.onRequestPermissionsResult(requestCode, permissions, grantResults);

 if (requestCode == ToneFramework.TONE_PERMISSION_CODE) {
   // Checking whether the user granted the permission or not.
   if (grantResults.length > 0 && (grantResults[0] + grantResults[1] ==
PackageManager.PERMISSION_GRANTED)) {
     // Start Service
     toneFramework.start();
   } else {
     Toast.makeText(MainActivity.this, "Permission Denied", Toast.LENGTH_SHORT).show();
   }
 }
}
```

End Service:

- The '**onDestroy**' method is overridden to ensure proper cleanup when the activity is destroyed.
- It calls the **finish** method of **toneFramework** to end the service.

```
  @Override
  public void onDestroy() {
    super.onDestroy();
    try{
      toneFramework.finish();
    }catch (IllegalArgumentException|NullPointerException e){
      e.printStackTrace();
    }
  }
 }
```

For Jetpack Compose:

- As the **'onDestroy'** method is not available in Jetpack compose, we can use the **'DisposableEffect'** method for proper cleanup, when composable is destroyed.
- It calls the **finish** method of **toneFramework** to end the service.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    toneFramework = ToneFramework(this)
    toneFramework.checkPermission(ToneFramework.TONE_PERMISSION_CODE, this)
    toneFramework.setClientId(clientId)
    setContent {
        SampleToneComposeTheme {
            Surface(
                modifier = Modifier.fillMaxSize(),
                color = MaterialTheme.colorScheme.background
            ) {
                Greeting("Tone Framework")
                CleanUpTask()
            }
        }
    }
}

@Composable
private fun CleanUpTask() {
    DisposableEffect(key1 = Unit) {
        onDispose {
            toneFramework.finish()
        }
    }
}
```

Restart Service:

- The '**restart**' method of '**toneFramework**' is called to restart the service.
- For Example:
  `toneFramework.restart();`

Tone Received:

- The '`onToneReceived`' method is implemented as part of the '`ToneUIEventListener`' interface.

- When a tone is received, this method is called, and it delegates the handling of the tone to the **toneFramework** by calling **handleToneResponse**.

```
@Override
public void onToneReceived(ToneModel toneModel) {
     toneFramework.handleToneResponse(toneModel, this);
}
```

Tone Tag:

- The '**onToneTag**' method is implemented as part of the '**ToneUIEventListener**' interface.
- When a ToneTag is detected, we can get that tone tag from this method.

```
@Override
void onToneTag(String toneTag) {
   Log.d("ToneTag", "ToneTag: $toneTag");
}
```

Bad Code:

- The '**onBadCode**' method is implemented as part of the '**ToneUIEventListener**' interface.
- We can get the detected code as good code or bad code as, boolean value from this method

```
@Override
void isBadCode(Boolean badCode) {
   if (badCode == true)
        Log.d("Tag Result", "Bad Code");
   else
        Log.d("Tag Result", "Good Code");
}
```

Handle Tone Response:

- The '**handleToneResponse()**' method can be called from toneFramework.

- This method is responsible for handling tone response, when a tone is received.
- We can pass '**`ToneModel`**' pass and context as params.
- From this method, we can handle the response from API, we can show data to the user Using this method.
- Usually call this method inside '**`onToneReceived()`**' method and pass the '**`toneModel`**' that we get from onToneReceived and pass it as a parameter to '**`handleToneResponse()`**'.

```
@Override
public void onToneReceived(ToneModel toneModel) {
 toneFramework.handleToneResponse(toneModel, this@MainActivity);
}
```

Handle Intent:

- '**`handleIntent()`**' is a method that can be called from '**`ToneFramework`**' as `toneFramework.handleIntent()`.
- This method accepts intent as a parameter when tone action detected.
- The action of the intent should match the constant `AppConstants`.TONE_DETECTED_ACTION.
- Create an Intent with actionType and actionUrl as key.
- When we pass intent as params, it will create a toneModel, and pass it to the `handleToneResponse()` method.

```
Intent intent = new Intent();
intent.setAction(AppConstants.TONE_DETECTED_ACTION);
intent.putExtra("actionType", "image");
intent.putExtra("actionUrl", "https://example.com/image.jpg");
toneFramework.handleIntent(intent);
```

Service running in foreground:

- The `isServiceRunningInForeground()` is a static method that can be called from ToneFramework.
- This method takes context and class as its two params.
- This method returns a non-nullable boolean value.
- If the ToneFramework service is running in foreground it returns true otherwise false.

```
ToneFramework.isServiceRunningInForeground(this, MainActivity.class);
```

SetToneTagInterface:

- The `setToneTagInterface()` is a method that can be called from ToneFramework.
- This method is used to set a Tone Tag Interface for the Tone Framework. It takes a functional interface as an argument.

```
toneFramework.setToneTagInterface(toneTag -> Log.d("TAG", "setToneTag: "+toneTag));
```

Proguard Rules:

- To add proguard-rules, Navigate to the app directory in your root directory.
- Inside the app directory you can find proguard-rules.
- If your project contains Webview with Javascript, then add this proguard-rule.

```
    # Add project specific ProGuard rules here.
```

```
# You can control the set of applied configuration files using the
# proguardFiles setting in build.gradle.
#
# For more details, see
#   http://developer.android.com/guide/developing/tools/proguard.html

# If your project uses WebView with JS, uncomment the following
# and specify the fully qualified class name to the JavaScript interface
# class:
#-keepclassmembers class fqcn.of.javascript.interface.for.webview {
#   public *;
#}

# Uncomment this to preserve the line number information for
# debugging stack traces.
#-keepattributes SourceFile,LineNumberTable

# If you keep the line number information, uncomment this to
# hide the original source file name.
#-renamesourcefileattribute SourceFile
```

Reducing app size:

Uploading your app using app bundle format will reduce the size of your app. The app bundle format uses the Split APK mechanism which allows Google Play to break up a large app into smaller, discrete packages that are installed on a user's device as required. Split APKs are very similar to regular APKs — they include compiled DEX bytecode, resources, and an Android

manifest. However, the Android platform is able to treat multiple installed split APKs as a single app.

Steps to build an app bundle:

You can easily build your app bundle using Android Studio(3.2 Canary 14+)

Go to **Build** > **Build Bundle(s)/APK(s)** and select **Build Bundle(s)**

or use the below command in the Android studio CLI console

./gradlew bundle

The generated app bundle will be stored at app/build/outputs/bundle/buildVariant/bundle.aab.