Wellington Cesar Fonseca

DESENVOLVIMENTO FULL STACK

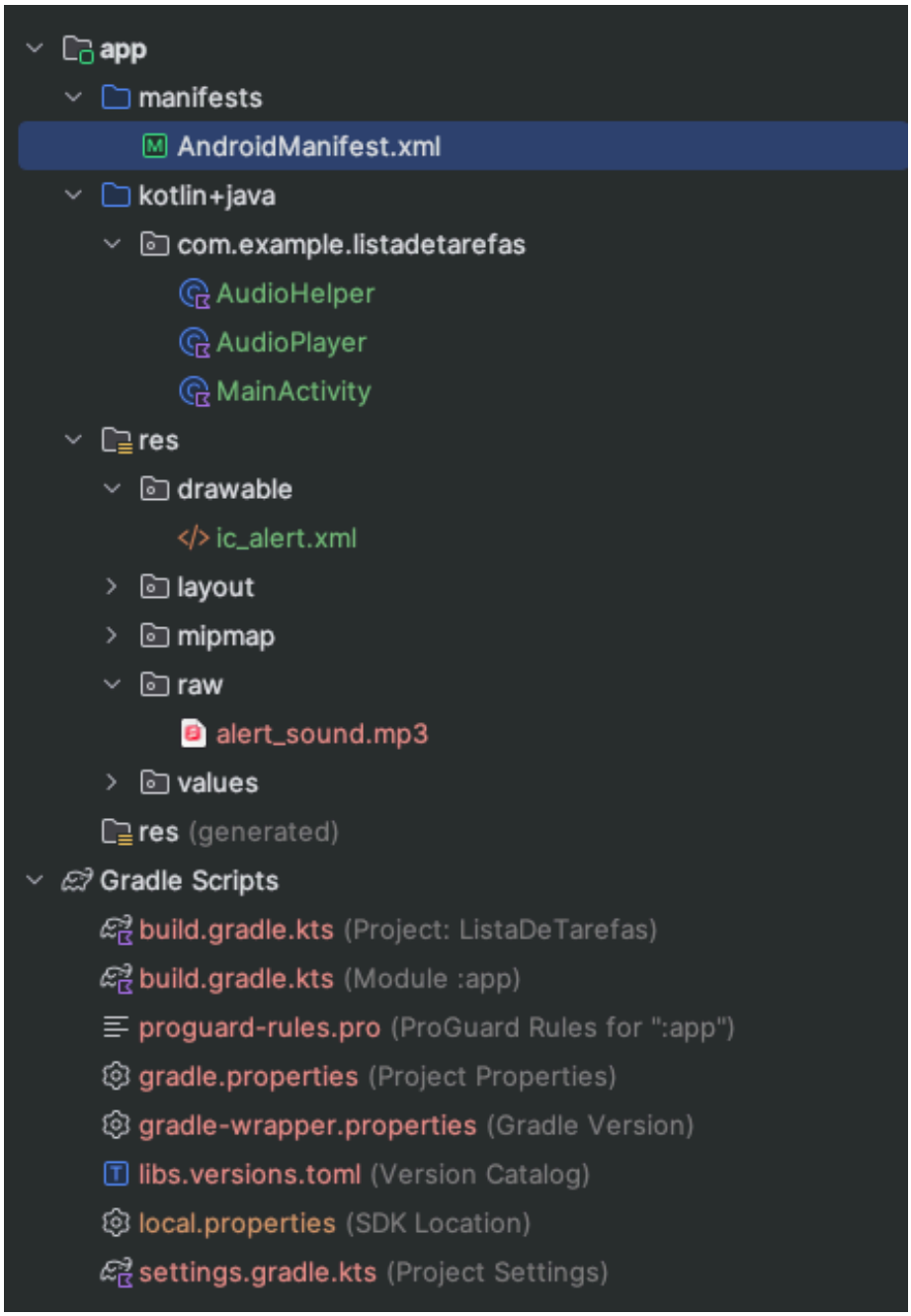
MISSÃO PRÁTICA | NÍVEL 3 | MUNDO 4

RPG0025 - LIDANDO COM SENSORES EM DISPOSITIVOS MÓVEIS

OBJETIVO

1. Instalação do Android Studio e do emulador;
2. Criar um app para Wear OS;
3. Executar um app no emulador;
4. Fazer capturas de telas no Android Studio;
5. Fazer capturas de telas com app complementar.

MISSÃO PRÁTICA | VAMOS CRIAR UM APP!



IMPORTANTE!

Não consegui fazer o código funcionar, recorri ao ChatGPT e também não deu certo. Ao questionar a diferença dos códigos ao GPT obtive o seguinte:

Análise das Alterações:

1. Funcionalidade de Áudio Original
2. A implementação inicial fornecida na orientação abordava a detecção de saídas de áudio, como alto-falantes internos e Bluetooth. Também incluía callbacks para detectar mudanças dinâmicas nos dispositivos conectados.
3. Alterações feitas no código atual:
   - O código original foi adaptado para verificar os dispositivos de áudio e exibir logs no Logcat com os resultados.
   - Foi adicionada uma funcionalidade básica de reprodução de áudio (AudioPlayer), que não estava presente nas orientações iniciais.
4. Notificações
5. Problema na orientação inicial:
   - A orientação original não mencionava a necessidade de permissões explícitas para notificações no Android 13+ (POST_NOTIFICATIONS).
   - No código atual, adicionamos solicitações de permissão e verificações explícitas para garantir que as notificações sejam exibidas.
   - Logs foram inseridos para monitorar o status da permissão e a tentativa de exibir notificações.
6. Reconhecimento de Voz
7. Adição no código atual:
   - A funcionalidade de reconhecimento de voz com entrada de comandos foi completamente adicionada no código atual, embora não estivesse mencionada na orientação inicial.
   - Essa inclusão amplia as funcionalidades e atende à proposta de comunicação por comandos de voz.
8. Botão para Testes
9. Inclusão no código atual:
   - Foi implementado um botão para iniciar o reconhecimento de voz, facilitando testes manuais.
   - Isso simplifica a interação para validar a funcionalidade diretamente na interface do app.

Resumo das Diferenças

1. Áudio – Adaptamos a verificação de dispositivos, adicionamos logs e incluímos a reprodução de som como exemplo.
2. Notificações – Implementamos permissões explícitas para Android 13+ e adicionamos logs para monitorar o processo.
3. Reconhecimento de Voz – Incluímos funcionalidade de comandos por voz, aprimorando o escopo original.
4. Interatividade – Adicionamos um botão para facilitar os testes sem precisar alterar o código.

Conclusão

Embora a implementação inicial fosse funcional para o áudio, faltava suporte para permissões modernas e notificações no Android 13+. Também não contemplava comandos de voz, algo que o código atualizado cobre.

Se desejar simplificar o código ou remover alguma dessas funcionalidades adicionais, podemos ajustar conforme sua necessidade.

app/manifests/AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">

    <!-- Identifica que o app é para Wear OS -->
    <uses-feature android:name="android.hardware.type.watch" />

    <!-- Permissões necessárias -->
    <uses-permission android:name="android.permission.BODY_SENSORS"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
    <uses-permission android:name="android.permission.RECORD_AUDIO"/>
    <uses-permission android:name="android.permission.INTERNET"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@android:style/Theme.DeviceDefault">

        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:theme="@android:style/Theme.DeviceDefault.NoActionBar">

            <!-- Intent-filter necessário -->
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

        </activity>
    </application>

</manifest>
```

app/kotlin+java/com.example.listadetarefas/AudioHelper.kt

```
package com.example.listadetarefas

import android.content.Context
import android.media.AudioDeviceInfo
import android.media.AudioManager
import android.media.AudioDeviceCallback
import android.media.AudioManager.GET_DEVICES_OUTPUTS
import android.content.pm.PackageManager
import android.util.Log

class AudioHelper(private val context: Context) {

    private val audioManager: AudioManager =
        context.getSystemService(Context.AUDIO_SERVICE) as AudioManager

    // Verifica se a saída de áudio está disponível
    fun audioOutputAvailable(type: Int): Boolean {
        if (!context.packageManager.hasSystemFeature(PackageManager.FEATURE_AUDIO_OUTPUT)) {
            return false
        }

        return audioManager.getDevices(GET_DEVICES_OUTPUTS).any { it.type == type }
    }

    // Callback para mudanças nos dispositivos de áudio
    fun registerAudioDeviceCallback() {
        audioManager.registerAudioDeviceCallback(object : AudioDeviceCallback() {

            // Dispositivo Conectado
            override fun onAudioDevicesAdded(addedDevices: Array<out AudioDeviceInfo>?) {
                addedDevices?.forEach {
                    Log.d("AudioCallback", "Dispositivo conectado: ${it.type}")
                }
            }

            // Dispositivo Desconectado
            override fun onAudioDevicesRemoved(removedDevices: Array<out AudioDeviceInfo>?) {
                removedDevices?.forEach {
                    Log.d("AudioCallback", "Dispositivo desconectado: ${it.type}")
                }
            }

        }, null)
    }
}
```

app/kotlin+java/com.example.listadetarefas/AudioPlayer.kt

```
package com.example.listadetarefas

import android.content.Context
import android.media.MediaPlayer

class AudioPlayer(private val context: Context) {

    private var mediaPlayer: MediaPlayer? = null

    // Iniciar a reprodução de áudio
    fun playAudio() {
        // Se houver uma reprodução ativa, pare antes de iniciar outra
        stopAudio()

        // Carregar o som da pasta raw
        mediaPlayer = MediaPlayer.create(context, R.raw.alert_sound)
        mediaPlayer?.start()
    }

    // Parar a reprodução de áudio
    fun stopAudio() {
        mediaPlayer?.apply {
            if (isPlaying) {
                stop()
                release()
            }
        }
        mediaPlayer = null
    }
}
```

app/kotlin+java/com.example.listadetarefas/MainActivity.kt

```
package com.example.listadetarefas

import android.app.Activity
import android.app.NotificationChannel
import android.app.NotificationManager
import android.app.PendingIntent
import android.content.ActivityNotFoundException
import android.content.Intent
import android.content.pm.PackageManager
import android.media.AudioDeviceInfo
import android.media.RingtoneManager
import android.os.Build
import android.os.Bundle
import android.speech.RecognizerIntent
import android.widget.Button
import android.widget.Toast
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.core.app.NotificationCompat
import androidx.core.app.NotificationManagerCompat
import androidx.core.content.ContextCompat
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import android.Manifest
import java.util.Locale

class MainActivity : AppCompatActivity() {
    private val REQUEST_CODE_SPEECH_INPUT = 100
    private val NOTIFICATION_REQUEST_CODE = 101

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_main)

        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
            insets
        }

        // Verificar permissão de notificação
        android.util.Log.d(
            "PermissionCheck",
            "Permissão POST_NOTIFICATIONS: " +
                    (ContextCompat.checkSelfPermission(
                        this, Manifest.permission.POST_NOTIFICATIONS
                    ) == PackageManager.PERMISSION_GRANTED)
        )

        val audioHelper = AudioHelper(this)

        // Verifica se há alto-falante disponível
        val isSpeakerAvailable = audioHelper.audioOutputAvailable(AudioDeviceInfo.TYPE_BUILTIN_SPEAKER)
        val isBluetoothHeadsetConnected = audioHelper.audioOutputAvailable(AudioDeviceInfo.TYPE_BLUETOOTH_A2DP)

        // Exibe no Logcat os resultados
        android.util.Log.d("AudioCheck", "Alto-falante disponível: $isSpeakerAvailable")
        android.util.Log.d("AudioCheck", "Fone Bluetooth conectado: $isBluetoothHeadsetConnected")

        // Registra o callback para monitorar mudanças nos dispositivos de áudio
        audioHelper.registerAudioDeviceCallback()

        // Solicita permissão para notificações antes de exibir
        requestNotificationPermission()

        ActivityCompat.requestPermissions(
            this,
            arrayOf(Manifest.permission.POST_NOTIFICATIONS),
            NOTIFICATION_REQUEST_CODE
        )

        // Botão de teste para entrada de voz
        val addTaskButton = findViewById<Button>(R.id.add_task_button)
        addTaskButton.setOnClickListener {
            startVoiceInput()
        }

        val audioPlayer = AudioPlayer(this)
        audioPlayer.playAudio()
    }

    // Solicitação de permissão para notificações
    private fun requestNotificationPermission() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
            if (ContextCompat.checkSelfPermission(
                    this, android.Manifest.permission.POST_NOTIFICATIONS
                ) != PackageManager.PERMISSION_GRANTED
            ) {
                ActivityCompat.requestPermissions(
                    this,
                    arrayOf(android.Manifest.permission.POST_NOTIFICATIONS),
                    NOTIFICATION_REQUEST_CODE
                )
            } else {
                // Permissão já concedida
                showNotification("Alerta Importante!", "Este é um alerta de exemplo.")
            }
        } else {
            // Versões anteriores não exigem permissão
            showNotification("Alerta Importante!", "Este é um alerta de exemplo.")
        }
    }

    // Função para exibir notificações
    private fun showNotification(title: String, message: String) {
        android.util.Log.d("NotificationCheck", "Tentando exibir notificação")

        val channelId = "alert_channel"

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val channel = NotificationChannel(
                channelId,
                "Alertas",
                NotificationManager.IMPORTANCE_HIGH
            ).apply {
                description = "Canal para alertas importantes"
            }

            val notificationManager: NotificationManager =
                getSystemService(NotificationManager::class.java)
            notificationManager.createNotificationChannel(channel)
        }

        val intent = Intent(this, MainActivity::class.java).apply {
            flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
        }
        val pendingIntent: PendingIntent = PendingIntent.getActivity(
            this, 0, intent, PendingIntent.FLAG_IMMUTABLE
        )
```

```
        val notification = NotificationCompat.Builder(this, channelId)
            .setSmallIcon(android.R.drawable.ic_dialog_alert)
            .setContentTitle(title)
            .setContentText(message)
            .setPriority(NotificationCompat.PRIORITY_HIGH)
            .setContentIntent(pendingIntent)
            .setAutoCancel(true)
            .setSound(RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION))
            .build()

        NotificationManagerCompat.from(this).notify(1, notification)
    }

    // Processar resultado da permissão
    override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<out String>,
        grantResults: IntArray
    ) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults)
        if (requestCode == NOTIFICATION_REQUEST_CODE) {
            if (grantResults.isNotEmpty() && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                showNotification("Alerta Importante!", "Este é um alerta de exemplo.")
            } else {
                Toast.makeText(this, "Permissão negada!", Toast.LENGTH_SHORT).show()
            }
        }
    }

    // Função para iniciar reconhecimento de voz
    private fun startVoiceInput() {
        val intent = Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH)
        intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.LANGUAGE_MODEL_FREE_FORM)
        intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, Locale.getDefault())
        intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "Fale algo...")

        try {
            startActivityForResult(intent, REQUEST_CODE_SPEECH_INPUT)
        } catch (e: ActivityNotFoundException) {
            Toast.makeText(this, "Dispositivo não suporta reconhecimento de voz", Toast.LENGTH_SHORT).show()
        }
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)

        if (requestCode == REQUEST_CODE_SPEECH_INPUT && resultCode == Activity.RESULT_OK) {
            val result = data?.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS)
            val recognizedText = result?.get(0)

            android.util.Log.d("VoiceInput", "Texto reconhecido: $recognizedText")
            Toast.makeText(this, "Reconhecido: $recognizedText", Toast.LENGTH_LONG).show()
        }
    }
}
```

Grandle Scripts/build.gradle.kts (Module :app)

```
plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.kotlin.android)
}

android {
    namespace = "com.example.listadetarefas"
    compileSdk = 34

    defaultConfig {
        applicationId = "com.example.listadetarefas"
        minSdk = 30
        targetSdk = 34
        versionCode = 1
        versionName = "1.0"

    }

    buildTypes {
        release {
            isMinifyEnabled = false
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }
    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_1_8
        targetCompatibility = JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = "1.8"
    }
}

dependencies {

    implementation(libs.play.services.wearable)
    implementation(libs.appcompat)
    implementation(libs.material)
    implementation(libs.activity)
    implementation(libs.constraintlayout)
    // Novas dependências adicionadas no catalog
    implementation(libs.core.ktx)
    implementation(libs.wear)
    implementation(libs.wearable)
    implementation(libs.media)

}
```

Grandle Scripts/libs.versions.toml (Version Catalog)

```
[versions]
agp = "8.6.0"
kotlin = "1.9.0"
playServicesWearable = "18.0.0"
appcompat = "1.6.1"
material = "1.10.0"
activity = "1.8.0"
constraintlayout = "2.1.4"

[libraries]
play-services-wearable = { group = "com.google.android.gms", name = "play-services-wearable", version.ref = "playServicesWearable"
}
appcompat = { group = "androidx.appcompat", name = "appcompat", version.ref = "appcompat" }
material = { group = "com.google.android.material", name = "material", version.ref = "material" }
activity = { group = "androidx.activity", name = "activity", version.ref = "activity" }
constraintlayout = { group = "androidx.constraintlayout", name = "constraintlayout", version.ref = "constraintlayout" }
core-ktx = "androidx.core:core-ktx:1.9.0"
wear = "androidx.wear:wear:1.2.0"
wearable = "com.google.android.support:wearable:2.9.0"
media = "androidx.media:media:1.6.0"

[plugins]
android-application = { id = "com.android.application", version.ref = "agp" }
kotlin-android = { id = "org.jetbrains.kotlin.android", version.ref = "kotlin" }
```