

Dallamanalízis dokumentáció

Fekete Dávid, Battai István, Ádám Krisztián

2025

1. A projekt célja

A Dallamanalízis egy olyan érintés- és hangalapú interaktív IoT rendszer, amelynek célja a felhasználó által létrehozott dallamok valós idejű érzékelése, feldolgozása és továbbítása egy mobilalkalmazás felé.

A rendszer érintés- és mozgásalapú bemeneteket, valamint digitális hangkimenetet egyesít egy hordozható, akkumulátoros eszközben.

2. Rendszer funkcionális céljai

2.1 Érintésalapú dallamvezérlés

több darab kapacitív érintésérzékelő:

- a hangsípek vagy dallamhangok aktiválását/vezérlését szolgálja, lehetővé teszi, hogy a felhasználó „játsszon” a rendszerrel, minden érintés külön bemeneti csatornaként működik.

2.2 Hanggenerálás I2S DAC-on keresztül

PCM5102A DAC feladata:

- a mikrokontroller által feldolgozott digitális hangadatok analóg audio jellé alakítása, így az eszköz valós időben hallható hangot képes generálni a dallam alapján, támogatja a frekvenciaalapú dallamképzést, skálák, hangmagasságok kezelését.

2.3 Dallam és frekvenciaadatok BLE továbbítása

Az ESP32:

- összegyűjti a dallam- és frekvenciaadatokat,
- a Dallamanalízis projekt saját BLE protokollján keresztül továbbítja az Androidalkalmazás felé,
- élő paramétereket küld:
 - aktuális MIDI frekvencia,
 - A4 referencia (kalibráció),
 - kontra síp állapot, ◦
akkumulátor töltöttség.

2.4 Android alapú élő visszajelzés és elemzés

A mobilalkalmazás valós időben:

- megjeleníti a bejövő hangfrekvenciát,
- jelzi a szenzorok állapotát, kezeli a hangerőket / síptípusokat, elemezheti a dallamot.

3. Hardveres célok

- hordozható, újratölthető 18650 akkumulátorral működő eszköz,
 - stabil 3.3V tápellátással minden komponens számára,
 - több érintésérzékelő és egy MPU6050 mozgásérzékelő integrálása, audio kimeneti lánc felépítése I2S protokollon.
-

4. Rendszer összefoglalása egy mondatban

A Dallamanalízis egy multimodális IoT hangszer és elemző rendszer, amely érintésérzékeléssel és digitális hangfeldolgozással generál dallamokat, majd ezek paramétereit valós időben továbbítja egy Android-alkalmazásnak elemzés és megjelenítés céljából.

1. Rendszer Áttekintése

Az IoT rendszer több szenzoros bemeneti modult és egy BLE-képes mikrokontrollert tartalmaz, amely valós időben továbbítja az adatokat egy Android-alkalmazás felé. A fő funkció: érintésérzékelők és/vagy nyomógombok által generált események Bluetooth Low Energy (BLE) kapcsolaton keresztül továbbítása mobilra. **A rendszer részei:**

- Mikrokontroller (Arduino/ESP32 vagy hasonló) – központi vezérlés
- CAPACITIVE TOUCH/TTP223 érintésérzékelők (8–10 db)
- MPU6050 gyorsulásmérő/gyroszkóp modul – mozgásadatok
- USB-UART / Tápmodul – energiabiztosítás
- Android alkalmazás – BLE kommunikáció, adatfogadás, UI megjelenítés

2. Hardver Architektúra

2.1 Rendszerblokkok

Érzékelőréteg:

Több db TTP223
érintésérzékelő modul;
Minden modul digitális
(HIGH/LOW) kimenetet ad a
központi MCU-nak;
5V tápellátásról

működnek **Mozgásérzékelő réteg:**

MPU6050 I2C buszon csatlakozik:

- ✦ SDA → MCU SDA
 - ✦ SCL → MCU SCL
- 3 tengelyű gyorsulás + 3 tengelyű gyroszkóp adatok

Vezérlőréteg:

Arduino/ESP32 fejlesztőpanel Feladatok:

- ✦ érintésérzékelők olvasása
- ✦ MPU6050 adatok olvasása
- ✦ adatok csomagolása és BLE-n továbbítása
- ✦ kapcsolatmenedzsment (connect / disconnect / notify)

Kommunikációs réteg:

Bluetooth Low Energy 4.0+

Szolgáltatások:

- ✦ Custom BLE Service
- ✦ Custom BLE Characteristic (NOTIFY) **Mobil**

alkalmazás réteg: Android Kotlin alkalmazás Funkciók:

- ✦ BLE eszközkeresés

- ✦ Kapcsolódás saját Service UUID alapján
 - ✦ Adatok feldolgozása és megjelenítése
 - ✦ Kapcsolatkezelés (connect/watch/terminate)
-

3. Breadboard Bekötési Dokumentáció

3.1 Tápellátás

- A teljes rendszer 5V-ról működik (USB vagy külső tápmodul).
- A föld (GND) minden modul között közösítve van.

3.2 Érintésérzékelők bekötése

- Mindegyik TTP223 modul 3 tűvel rendelkezik: VCC, GND, OUT
- A VCC → 5V pontról ágazik
- A GND → közös földre
- Az OUT vezetékek a mikrokontroller különböző digitális pineire vannak bekötve Példa kiosztás (illusztratív):
 - ✦ Touch1 → D2
 - ✦ Touch2 → D3
 - ✦ ...
 - ✦ Touch8 → D9

3.3 MPU6050 bekötése

- SDA → MCU SDA (pl. pin A4 / GPIO21)
- SCL → MCU SCL (pl. pin A5 / GPIO22)
- VCC → 3.3V vagy 5V (modultól függően)
- GND → közös föld

3.4 USB-UART modul

- Biztosítja a tápot és a soros kommunikációs lehetőséget fejlesztéshez
 - 5V → breadboard tápvonal
 - GND → közös föld
-

4. Firmware Architektúra (Mikrokontroller)

4.1 Fő modulok

- **SensorManager**
 - Olvassa a TTP223 touch szenzorok és az MPU6050 értékeit.
 - **BLEManager**
 - Custom Service UUID
 - Characteristic (NOTIFY mód)
 - Kapcsolatfelépítés kezelése
 - **Main Loop**
 - Érzékelőadatok beolvasása ciklikusan
 - Változás esetén BLE értesítés küldése
 - Kapcsolatfigyelés (disconnect detection)
-

5. Android Alkalmazás Architektúra

5.1 Layer-szerkezet • UI Layer

- ✦ Gombok: scan, connect, watch, terminate

- ✦ TextView: aktuális BLE adatok megjelenítése
- **Service Layer**
 - ✦ BluetoothService – teljes BLE kommunikáció
 - ✦ eszközkeresés
 - ✦ kapcsolódás
 - ✦ characteristic olvasás/írás
 - ✦ notify kezelése
- **Data Layer**
 - ✦ BLE-ből érkező bitek → értelmezett szenzoradatok ✦ Feldolgozás és megjelenítés

5.2 Adatfolyam

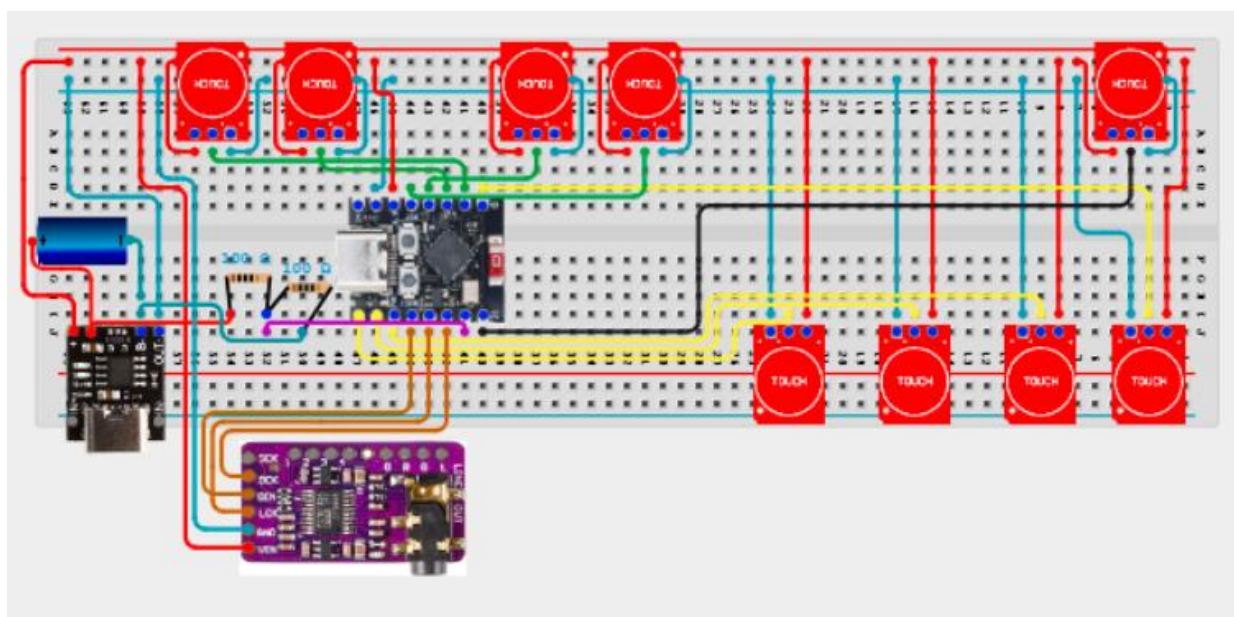
1. Felhasználó → "Scan"
2. App → BLE eszközlista
3. Felhasználó kiválasztja az IoT eszközt
4. "Connect" → BLE kapcsolat létrejön
5. App bekapcsolja a NOTIFY módot
6. MCU → érintés és mozgásadatokat küld
7. App → feldolgozás + UI frissítés

6. Adatformátum és Protokoll

6.1 BLE karakterisztika adatai A firmware a következőket küldi egy DATA csomagban:

- Touch szenzorok állapota (bitmező)
- Gyorsulási adatok (x, y, z)
- Giroszkóp adatok (x, y, z)
- Opcionálisan timestamp

Példa formátum:
[TOUCH_FLAGS][ACC_X][ACC_Y][ACC_Z][GYRO_X][GYRO_Y][GYRO_Z]



Az mobil applikáció működése

MainActivity működése

Ez a fejezet a projekt Android-oldali vezérlőmodulját, a MainActivity működését, szerepét és a komponensek együttműködését dokumentálja. A leírás célja, hogy átláthatóvá tegye a BLEkapcsolat kezelésének folyamatát, a felhasználói felülettel (UI) való integrációt, valamint az ESP32-eszközzel folytatott kommunikációt.

1. A MainActivity szerepe

A MainActivity az alkalmazás központi vezérlője.

Feladatai:

- A felhasználói felület inicializálása és frissítése.
- Bluetooth LE kapcsolat létrehozása az ESP32 eszközzel.
- A MainViewModel megfigyelése és a LiveData-változások kezelése.
- A BLE által biztosított GATT szolgáltatásokhoz való hozzáférés.
- Tranziensek, hibaállapotok és visszajelzések megjelenítése. A logikai működés

három fő rétegre oszlik:

1. **UI szint** (TextView, SeekBar, Button, ToggleButton)
2. **Állapotkezelés** (MainViewModel)
3. **Kommunikációs réteg** (BluetoothGattCallback)

2. Főbb alkalmazott technológiák

- **Android BLE API (BluetoothGatt)**

Az ESP32-vel való kapcsolatot GATT-alapú kommunikáció valósítja meg.

- **ViewModel + LiveData**

A megjelenített adatok megfigyelhető, eseményvezérelt módon frissülnek.

- **UI vezérlők**

Hangerőszabályzó csúszkák, állapotkijelzők, kapcsolók, log üzenetek.

3. Adatfolyam (ESP32 → Android → UI)

1. ESP32

Küldi a BLE-adatcsomagot →

2. Android (GattCallback)

Az onCharacteristicChanged() feldolgozza a ByteArray-t → továbbítja a ViewModel felé →

3. ViewModel

Feldolgozza a protokoll szerint:

- MIDI frekvencia
- Kontra síp állapot
- Akkumulátor töltöttség

→ LiveData frissül →

4. UI komponensek (MainActivity)

Az observe() függvények alapján automatikusan módosulnak.

4. BLE kapcsolatkezelés

A MainActivity az ESP32 hirdetett eszközére keres rá.

A kapcsolat állapota a MainViewModel.connectionState LiveData-ban jelenik meg.

A BLE kommunikáció az alábbi lépésekben zajlik:

1. Bluetooth Adapter lekérése
2. Eszköz keresés és beolvasás
3. Kapcsolódás (connectGatt)
4. Szolgáltatások lekérése (discoverServices)
5. Karakterisztikák kiolvasása / iratkozás (notify)
6. **Élő adatok fogadása**

A dokumentált protokoll szerint 4 bájtos csomag érkezik.

5. UI-komponensek dokumentációja

5.1. Állapot kijelzés

- Bluetooth kapcsolat szöveges visszajelzése
- Akkumulátor töltöttség megjelenítése
- MIDI frekvencia és referenciahang kijelzés

5.2. Felhasználói bemenetek

- Három hangerőszabályzó csúszka (Melody / Kontra / Bordó)
- Kontra mód kapcsoló (ToggleButton)

A felhasználó által végzett műveletek BLE karakterisztika-írást indítanak.

6. BLE karakterisztika írások dokumentációja

A felhasználó módosításai (pl. hangerő vagy kontra állapot) karakterisztikába íródnak. Az adatküldés sémája:

Funkció	Küldött bájt(ok)	Jelentés
Hangerő beállítás	[0x02, dVol, kVol, bVol]	Melody / Kontra / Bordó
Kontra mód	[0x03, 0x01] vagy [0x03, 0x00]	be / ki
A4 referenciafrekvencia beállítás egyedi parancs		(opcionális)

7. Hibakezelés

A MainActivity naplóban és vizuálisan is jelzi:

- BLE kapcsolat hiba
- Szolgáltatás / karakterisztika hiánya
- Engedély hiány
- GATT error kódok

A kommunikációs hibák nem omlasztják össze az alkalmazást: a ViewModel-en keresztül visszajelzést ad → az UI újrapróbál.

8. Lifecycle kezelés

A MainViewModel segítségével:

- a BLE állapot nem vész el képernyőforgatáskor,
- a LiveData biztosítja a kicsatolt, thread-safe állapotkezelést, • a MainActivity csak UI-t frissít, az adatlogika nem szóródik szét.

9. Összegzés

A MainActivity a rendszer központi vezérlője, amely:

- megnyitja és fenntartja a BLE kapcsolatot,
 - valós időben fogadja az ESP32 által küldött hang- és állapotadatokat,
 - LiveData segítségével dinamikusan frissíti a felhasználói felületet, • a felhasználói parancsokat BLE karakterisztika-írás formájában továbbítja,
 - stabil és jól strukturált IoT-kommunikációs réteget biztosít Androidon.
-

MainViewModel osztály

A MainViewModel osztály az Android alkalmazás állapotkezelő rétegét valósítja meg a Jetpack Architecture Components segítségével. Feladata, hogy az ESP32 mikrokontroller Bluetooth Low Energy (BLE) csatornáján érkező adatokat feldolgozza és a mobilalkalmazás UI komponensei számára megfigyelhető (LiveData) formában biztosítsa. A ViewModel biztosítja, hogy az alkalmazás állapota tartós maradjon az Activity konfigurációs változásai során (pl. képernyőelforgatás), és elkülöníti az alkalmazáson belüli adatlogikát a felhasználói felülettől.

Fő funkciók

1. Bluetooth kapcsolat állapotkezelése

- A connectionState: MutableLiveData<String> a BLE kapcsolat aktuális státuszát tárolja (pl. „KAPCSOLÓDVA”, „KERESÉS...”, „HIBA: 133”).
 - Ez az állapot közvetlenül megjelenik a UI-ban.
-

2. Kezdeti hangerőértékek kezelése A InitStatus adatmodell tartalmazza:

- pipe – kezdeti aktív síp (a BLE protokoll része),
- dVol, kVol, bVol – a három síp kezdeti hangerőértékei. A ViewModel privát MutableLiveData-t és publikus, megfigyelhető LiveData-t biztosít:

private val _initStatus = MutableLiveData<InitStatus>() val initStatus:

LiveData<InitStatus> = _initStatus A frissítéshez az updateInitStatus() metódus használható.

3. Élő szenzoradatok fogadása (ESP32 → Android)

A ViewModel a BLE eszköztől érkező adatcsomagokat a onDataReceived(data: ByteArray) függvényben dolgozza fel.

A használt adatprotokoll:

Byte index		Jelentés	Típus
0	MIDI frekvencia LSB	UInt8	
Byte index		Jelentés	Típus
1	MIDI frekvencia MSB	UInt8	
2	Kontrasíp állapot (0/1)	Bool	

- 3 Akkumulátor töltöttség (%) UInt8 **Feldolgozás:**
- **MIDI frekvencia:** 16 bites (Little-endian) érték → Float formátumban továbbítva.
 - **Kontra síp állapot:** logikai értékké alakítás.
 - **Akkumulátor töltöttség:** százalékos érték. A ViewModel a következő LiveData-kat frissíti:
- liveMidiFreq: MutableLiveData<Float>
 - liveKontraState: MutableLiveData<Boolean>
 - liveBattery: MutableLiveData<Int>
 - frequency: MutableLiveData<Int> (A4 referenciafrekvencia az ESP32-től)
-

BagpipeSynth működése

Ez a fejezet a projekt hanggeneráló alrendszerét dokumentálja, amely a duda három sípjának (dallam, kontra, bordó) digitális szintézisét, lejátszását és opcionális WAVfelvételét végzi. Az osztály valós időben dolgozik, külön háttérszálon, és közvetlenül PCM mintákat állít elő az Android AudioTrack számára.

1. Modul célja és szerepe

A BagpipeSynth osztály egy valós idejű hangszintetizátor, amely:

- három külön digitális oszcillátort generál (melody, kontra, bordó),
- ezek jeleit összekeveri és PCM formátumba alakítja,
- közvetlenül írja az Android hangkimeneti pufferébe (AudioTrack),
- igény szerint WAV felvételt készít a generált audióról.

A modul az Android alkalmazás és az ESP32 IoT-szenzoroldal között biztosítja az „akusztikus” réteget: az ESP32 által küldött frekvenciaadatokból tényleges hangot állít elő.

2. Fő funkciók áttekintése

A BagpipeSynth fő feladatcsoportjai:

1. Valós idejű oszcillátor alapú hangszer-szintézis

- Fűrészfog jelalak generálás három párhuzamos síphoz
- Oktáv és modulációs transzformációk Digitális keverés és hangerőkezelés
- Clipping-limitáció

2. AudioTrack-alapú kimenetkezelés

- Mono, 16 bit PCM, 44.1 kHz mintavétel
- Áramlásos (stream) üzemmód valós idejű lejátszáshoz
- Háttérszálon futó mintairás

3. WAV felvétel támogatása

- Nyers PCM adat írása fájlba
 - 44 bájtos WAV-headerek generálása és utólagos javítása
 - Little-endian adatsorrend kezelése
4. Szálbiztos állapotkezelés
- A fő frekvenciák és hangerők @Volatile jelöléssel
 - Biztonságos olvasás/írás a szintetizáló szál és az UI/Bluetooth logika között

3. Architektúra és adatáramlás

ESP32 → MainViewModel → BagpipeSynth → AudioTrack

→ Hangszóró A lépések:

1. **ESP32 BLE adatok → ViewModel** Frekvencia, kontra állapot, hangerők.
2. **ViewModel → BagpipeSynth**
 - A változások a @Volatile mezőkbe kerülnek.
3. **BagpipeSynth → AudioTrack**
 - Hanggeneráló szál fut 44.1 kHz mintavételezéssel
 - PCM adatpuffer kerül az AudioTrack-be
4. **WAV felvétel esetén**

A generált minták párhuzamosan fájlba is íródnak.

4. Audió szintézis részletes dokumentáció

4.1 Mintavételezési paraméterek

- **Mintavételi frekvencia:** 44,100 Hz
- **Csatorna:** Mono
- **Bitmélység:** 16 bit PCM

Ez CD-minőségű mintavételezés.

4.2 Sípok frekvenciamodellje

Síp	Levezetett frekvencia	Magyarázat
Bordó	$\text{baseFreq} / 4.0$	két oktávval mélyebb drone
Kontra	$\text{baseFreq} / 2.0$	egy oktávval mélyebb drone
Síp	Levezetett frekvencia	Magyarázat
Kontra modulált	$\text{baseFreq} / 2.0 * 0.749$	„duda kontra effekt”
Dallam	currentMelodyFreq	BLE-ből érkező MIDI

A moduláció és oktávtranszformációk valós időben, a mintaszintről történnek.

4.3 Hanggenerálás mechanizmusa

A szintetizátor mindhárom sínél a következő jelalakot generálja: **Fűrészfog hullámformula:**

$\text{sample} = (\text{phase} / \text{PI} - 1.0) * \text{volume}$

- A fázis 0 és 2π között fut.
- Egy inkrementum (incX) minden mintánál hozzáadódik.
- 2π után újrakezdődik.

Minta generálási lépések:

1. **Oszcillátor frissítése** (fázis előreléptetés)
2. **Fűrészfog minta előállítása**
3. **Relatív hangerők alkalmazása**
4. **Összegzés (mix)**
5. **Clipping:** mixed $\in [-1.0, 1.0]$
6. **Konverzió PCM short típusra**
7. **Puffer `audioTrack.write()` hívása**

A teljes folyamat ciklikusan fut egy dedikált háttérszámban.

5. **AudioTrack működésének dokumentációja**

A modul a **STREAM** módot használja:

```
audioTrack = AudioTrack(  
    AudioManager.STREAM_MUSIC,  
    sampleRate,  
    AudioFormat.CHANNEL_OUT_MONO,  
    AudioFormat.ENCODING_PCM_16BIT,    bufferSize,  
    AudioTrack.MODE_STREAM  
)
```

A `write()` metódus a generált mintát közvetlenül továbbítja a hangrendszernek.

6. **WAV-felvétel alrendszer**

A WAV szabvány 44 bájtos fejlécet igényel.

A modul működése: **1.**

Felvétel indítása:

- Létrejön a stájl: `recordingStream = FileOutputStream(file)`
- Kiír egy **44 bájtos placeholder fejlécrészt**
- A generált PCM adatok valós időben íródnak a fájlba.

2. Felvétel leállítása:

- A stream lezárása után a modul a fájl elejére visszaugrik.
- Kitölti a fejléc valós méretadatait:
 - RIFF chunk méret
 - „fmt” struktúra
 - „data” blokk valódi hossza Ez teszi lejátszhatóvá a WAV fájlt.

7. **Szálkezelés**

A modul biztonságosan kezeli a párhuzamos műveleteket:

- A szintetizátor adatai (`currentMelodyFreq`, `hangerők`, `flags`) **@Volatile** változók
 - A háttérszál addig fut, amíg `isRunning = true`
- Leállításkor:
 - `join()`-nal megvárja a szálat
 - leállítja az `AudioTrack`-et
 - felszabadítja az erőforrásokat

8. Fő API műveletek

Metódus	Funkció
start()	Hangszintetizátor és AudioTrack indítása
stop()	Szál és hangkimenet leállítása
startRecording(file)	WAV felvétel indítása
stopRecording()	Felvétel leállítása, header javítása

A fenti metódusok hívása UI-ból vagy ViewModelből végezhető.

9. Modul használata az alkalmazásban

A BagpipeSynth a ViewModel-ből kapja az adatokat:

- baseFreq ← A4 referenciahang
- currentMelodyFreq ← BLE MIDI frekvencia
- isKontraModulated ← Kontra állapot
- volDallam, volKontra, volBordo ← hangerőértékek

A MainActivity ezeket a változókat módosítja a felhasználó interakciója alapján.

10. Összegzés

A BagpipeSynth modul:

- Magas teljesítményű, valós idejű hangszintetizátort valósít meg
- Három síp (melody, kontra, drone) modellezi a duda hangzását
- Fűrészfog hullámon alapuló digitális oszcillátort alkalmaz
- Biztonságos és stabil AudioTrack-pipeline megoldással dolgozik
- WAV felvételi képességet biztosít
- Szálbiztos és nagy mértékben testreszabható

A modul az IoT projekt kritikus része: az ESP32 által közölt adatokból valódi zenei hangzást állít elő.

BleManager működése

1. Áttekintés

A BleManager komponens a teljes Bluetooth Low Energy kommunikációért felelős az Android alkalmazás és az ESP32-alapú duda között. Feladatai:

- BLE szkennelés és eszközkeresés
- Csatlakozás és újracsatlakozás
- GATT szolgáltatások és karakterisztikák kezelése
- Adatküldés az ESP32 felé
- Élő szenzoradatok fogadása (pl. furulyalyuk állapot, nyomás, hangerő stb.)
- A ViewModel frissítése (UI update)

A működés alapelve: az app megtalálja az „ESP32-Duda” nevű eszközt, csatlakozik hozzá, bekapcsolja az értesítéseket (notifications), majd folyamatosan fogadja és továbbítja az adatokat.

2. Fő komponensek

2.1 BLE UUID-ek

A BLE kommunikáció az alábbi egyedi UUID-eket használja:

Funkció	UUID
Service (ESP32-Duda szolgáltatás)	00001234-0000-1000-8000-00805f9b34fb
Characteristic (adatírás/olvasás)	0000abcd-0000-1000-8000-00805f9b34fb
Notification Descriptor (CCCD)	00002902-0000-1000-8000-00805f9b34fb

A karakterisztika:

- **írható** → vezérlőparancsok küldése
- **notifiable** → élő adatok érkeznek az ESP32-től

3. Folyamatok részletes leírása

3.1 Indítás: keresés + csatlakozás

A `startScanAndConnect()` metódus:

1. Ellenőrzi a Bluetooth-adaptert.
2. Elindítja a BLE-szkennelést.
3. 10 másodperc után:
 - leállítja a szkennelést,
 - jelzi, ha nem találta meg az eszközt.
4. Ha eszköz található: csatlakozik hozzá. A szkennelés csak az „ESP32-Duda” kezdetű nevű eszközökre szűr.

3.2 Csatlakozás menete

A `connectToDevice()` hívódik meg, amely:

- GATT-kapcsolatot hoz létre
- Modern eszközökön a **TRANSPORT_LE** módot kényszeríti
- Átadja a callbacket (lásd 5. fejezet) Sikeres kapcsolat után a rendszer:
 1. **discoverServices()** hívást indít
 2. Lekéri a szolgáltatást
 3. Lekéri a karakterisztikát
 4. Bekapcsolja a **notifications**-t
 5. Kész állapotba kerül → READY

3.3 Automatikus újracsatlakozás

Ha bármilyen hiba vagy bontás történik (pl. távolság, ESP32 újraindul):

- Azonnal:
 - Lezárja a GATT-kapcsolatot
 - Nullázza a referenciákat
- 2 másodperc múlva automatikusan újraindítja a szkennelést
Ez garantálja a stabil „állandó kapcsolatot”.

3.4 Adatküldés az ESP32 felé

A writeCommand(bytes: ByteArray) metódus:

- Modern Androidon (API 33+): közvetlen writeCharacteristic(bytes)
- Régi Androidon:
 - a karakterisztikába írja a value mezőt o
 - majd writeCharacteristic() hívással küld

Használata:

- ble.writeCommand("DALLAM_ON".toByteArray())

3.5 Adatfogadás az ESP32-től A BLE adatok érkezése két formában történhet (API verziótól függ):

- **3 paraméteres** onCharacteristicChanged(g, characteristic, value)
- **régi 2 paraméteres** onCharacteristicChanged(g, characteristic)

A kód mindkettőt támogatja.

Mindkét esetben a handleCharacteristicChanged(value) metódus fut le. A beérkező nyers bájtokat a ViewModel kapja meg:

vm.onDataReceived(value) A

ViewModel dolga:

- kottázás
- sípok hangerővezérlése
- grafikus visszajelzés • szintetizátor frekvenciák frissítése stb.

4. Állapotkezelés (ViewModel integráció)

A BleManager NEM tartalmaz logikát a GUI-hoz vagy a hanghoz.

Ehelyett állapotokat küld egy ViewModelnek:

Állapot	Jelentése
"KERESÉS..."	BLE eszköz szkennelése
"MEGTALÁLVA: XYZ"	Találat
"CSATLAKOZÁS..."	Csatlakozási fázis
Állapot	Jelentése
"KAPCSOLÓDVA"	GATT kapcsolat aktív
"READY"	Adatkommunikáció kész
"MEGSZAKADT"	Kapcsolatvesztés
"HIBA: ..."	Kritikus BLE hiba

Ez tiszta MVVM architektúrát eredményez.

5. BluetoothGattCallback – események

részletes működése A callback a BLE életciklus szíve.

5.1 Csatlakozás állapota hibák esetén:

- automatikus reconnect

- állapotjelzés a UI felé

5.2 Szolgáltatás felfedezése

- Lekéri a kért szolgáltatást
- Ha nincs → hibajelzés
- Lekéri a karakterisztikát
- Értesítések engedélyezése a CCCD descriptoron keresztül

5.3 Descriptor írás után

Ha a notifikáció sikeresen aktiválódik → READY

5.4 Adatfogadás (notification) A bejövő adat:

- naplózva van
- átadásra kerül: `vm.onDataReceived(value)`

6. Hibakezelés és robusztusság A kód külön figyelmet fordít:

- **status 133** hiba automatikus kezelésére
- Android-verziók közti write metódus különbségeire •
GATT erőforrások szabályos lezárására
- 2-féle `onCharacteristicChanged` overload támogatására •
A reconnect logika miatt a rendszer önjavító.

7. Biztonság és engedélyek A fájl tetején szerepel:

`@SuppressWarnings("MissingPermission")`

Azért szükséges, mert:

- minden BLE hívás külön engedélyt igényel
- ezek kezelése a **MainActivity-ben** történik
- a `BleManager` nem felel a permission-logikáért

8. Összegzés – szerepe az IoT Duda projektben

A `BleManager` a projekt kommunikációs gerince.

Megvalósítja:

- stabil kapcsolat az ESP32 dudu között
 - valós idejű adatfolyam fogadása
 - parancsok és frekvenciák küldése
 - automatikus hibaelhárítás A modul egésze úgy lett felépítve, hogy:
- ✓ folyamatos és késleltetésmentes legyen
 - ✓ magától helyreálljon kapcsolatvesztés esetén
 - ✓ Android 5.0 – 14.0 között működjön
 - ✓ minimális UI-logikát tartalmazzon (MVVM kompatibilis)