

Kliensalkalmazások

Android 6 – NFC, Bluetooth, Http, Retrofit, BR, Service, Permissions

2024. 05. 13.

Gazdi László

gazdi.laszlo@aut.bme.hu



Automatizálási és
Alkalmazott
Informatikai Tanszék

Miről volt szó az előző órán?

- Compose alapok
- Compose Layout-ok
- Modifier-ek
- Compose alapelvek
- Recomposition
- ViewModelNavigáció
- Dialógusok
- Listák
- Szálkezelés, coroutine-ok
- Flow-k
- View és Compose átjárhatóság



Tartalom

- Rövidtávú kommunikáció:
 - > NFC
 - > Bluetooth
- Internet alapú kommunikáció:
 - > TCP/IP, UDP
 - > HTTP(s) kapcsolatok kezelése
 - Retrofit
 - > Tipikus adatformátumok
- Broadcast Receiver
- Service
- Futási idejű engedélyek

Hálózati kommunikáció

Hálózati kommunikáció Android platformon

- „Rövid” távú kommunikáció
 - > NFC
 - > Nearby API
 - > Bluetooth
 - > WiFi Direct
- „Hosszabb távú” / internet alapú kommunikáció
 - > TCP Socket
 - > UDP
 - > HTTP (REST Services - Representational State Transfer)

Near Field Communication (NFC)

- Rövidtávú vezeték nélküli kommunikációs technológia
- <4cm távolságon belül működik
- NFC tag és mobil telefon közti kis méretű adatátvitel (payload)
- Mobil telefon és mobil telefon közti kis méretű adatátvitel
- NFC Forum által meghatározott formátum: NDEF (NFC Data Exchange Format)

NFC Tag-ek jellemzői

- Írható/olvasható/egyszer írható
- Komplex Tag-ek tartalmazhatnak matematikai műveleteket is és lehet külön kriptográfia hardver egységük autentikáció céljából
- Még bonyolultabb Tag-ek akár saját működési környezettel is rendelkezhetnek és egyszerűbb kód végrehajtására is alkalmasak

NFC Beam

- NDEF üzenet küldése két Android készülék között
- Nincs szükség készülék felderítésre
- A kapcsolat automatikusan felépül, ha két eszköz hatótávon belülre kerül
- Beépített alkalmazások is támogatják adatcserére (pl.: Browser, Névjegyzék, YouTube stb.)

NFC használat lépései

- Permission

```
> <uses-permission android:name="android.permission.NFC" />
```

- Minimum Android verzió

```
> <uses-sdk android:minSdkVersion="10" />
```

- NFC hardware ellenőrzése (így nem jelenik meg azon eszközök számára, amik nem támogatják az NFC-t)

```
> <uses-feature android:name=  
    "android.hardware.nfc" android:required="true" />
```

Android Bluetooth APIk

- Hagyományos Bluetooth kommunikáció
 - > Részletes Bluetooth API-k
- Bluetooth LE támogatás
 - > Bluetooth LE API

Bluetooth

- Bluetooth készülékek felderítése
- Helyi Bluetooth adapter-ek lekérdezése a párosított eszközökhöz
- RFCOMM csatorna kiépítése
- Kapcsolódás service discovery-n keresztül
- Adattovábbítás készülékek közt
- Több egyidejű kapcsolat kezelése
- Fontosabb osztályok:
 - > android.bluetooth csomag

Legfontosabb Bluetooth osztályok

- *BluetoothAdapter*: felderítés, párosítások lekérdezése, *BluetoothDevice* példányosítása, server socket létrehozás
- *BluetoothDevice*: távoli Bluetooth eszközt jelképezi
- *BluetoothSocket/BluetoothServerSocket*
- *BluetoothClass*: eszköz tulajdonságai (read-only)
- *BluetoothProfile*: profil jellemzői
- Chat példa:
 - > <https://github.com/android/connectivity-samples/tree/master/BluetoothChat>

Bluetooth low energy

- Android 4.3 (API Level 18)
- Bluetooth Low Energy; *central role* és *peripheral* APIk
- Felderítés és kommunikáció támogatása (karakterisztikák írása)
- Lényegesen kevesebb energiahasználat (telefon és eszköz oldalon).
- Tipikus eszközök:
 - > Közelség érzékelők, szívritmus szenzorok, eü. Kiegészítők, stb.

Nearby API

- Három fő építő elem
- Nearby Messages
 - > Egyszerű üzenet küldés és fogadás
 - > byte[] küldhető
- Nearby Connections
 - > Egyszerű peer-to-peer kapcsolat kiépítés
- Nearby Notifications
 - > Weboldal vagy alkalmazás beacon-hoz csatolása, értesítések megjelenítése
- További információk:
 - > <https://developers.google.com/nearby/>

TCP/IP Socket

- Szabványos *Socket* implementáció
- Jól ismert *java.net.Socket* osztály a kapcsolatok megnyitására
- *java.net.ServerSocket* osztály a bejövő kapcsolatok fogadására
 - > Localhoston az alkalmazások egymás közti kommunikációja is megoldható
- *InputStream* és *OutputStream* támogatás az adatok olvasására és írására

Socket példa

```
val socket = Socket("192.168.2.112", 8787)
val inputStream = socket.getInputStream()
val isr = InputStreamReader(
    inputStream,
    "UTF-8"
)
val resultBuffer = StringBuilder()
var inChar: Int
while ((inChar = isr.read()) != -1) {
    resultBuffer.append(inChar.toChar())
}
val result = resultBuffer.toString()
// result kezelése
// ...
inputStream.close()
socket.close()
```


UDP kommunikáció

- User Datagram Protocol (UDP)
- Gyors kommunikáció
- Az UDP nem biztosítja a csomag megérkezését
- Tipikusan, ha nem fontos a csomag elvesztése
- Példa használat:
 - > Valós idejű multimédia átvitel
 - > Játékok
 - > Stb.

UDP Androidon

- Standard Java osztályok
- *java.net.DatagramSocket*: socket
- Broadcast támogatás:

```
DatagramSocket s = DatagramSocket()  
s.setBroadcast(true)
```
- *java.net.DatagramPacket*: UDP csomag

UDP üzenet küldése

```
val msg = "UDP Test"  
val socket = DatagramSocket()  
// In case of broadcast  
socket.setBroadcast(true)  
//InetAddress localAddr =  
//  InetAddress.getByName("192.168.0.110");  
val message = msg.toByteArray()  
val p = DatagramPacket(  
    message,  
    msg.length, localAddr, 10100  
)  
socket.send(p)
```

UDP üzenet fogadása

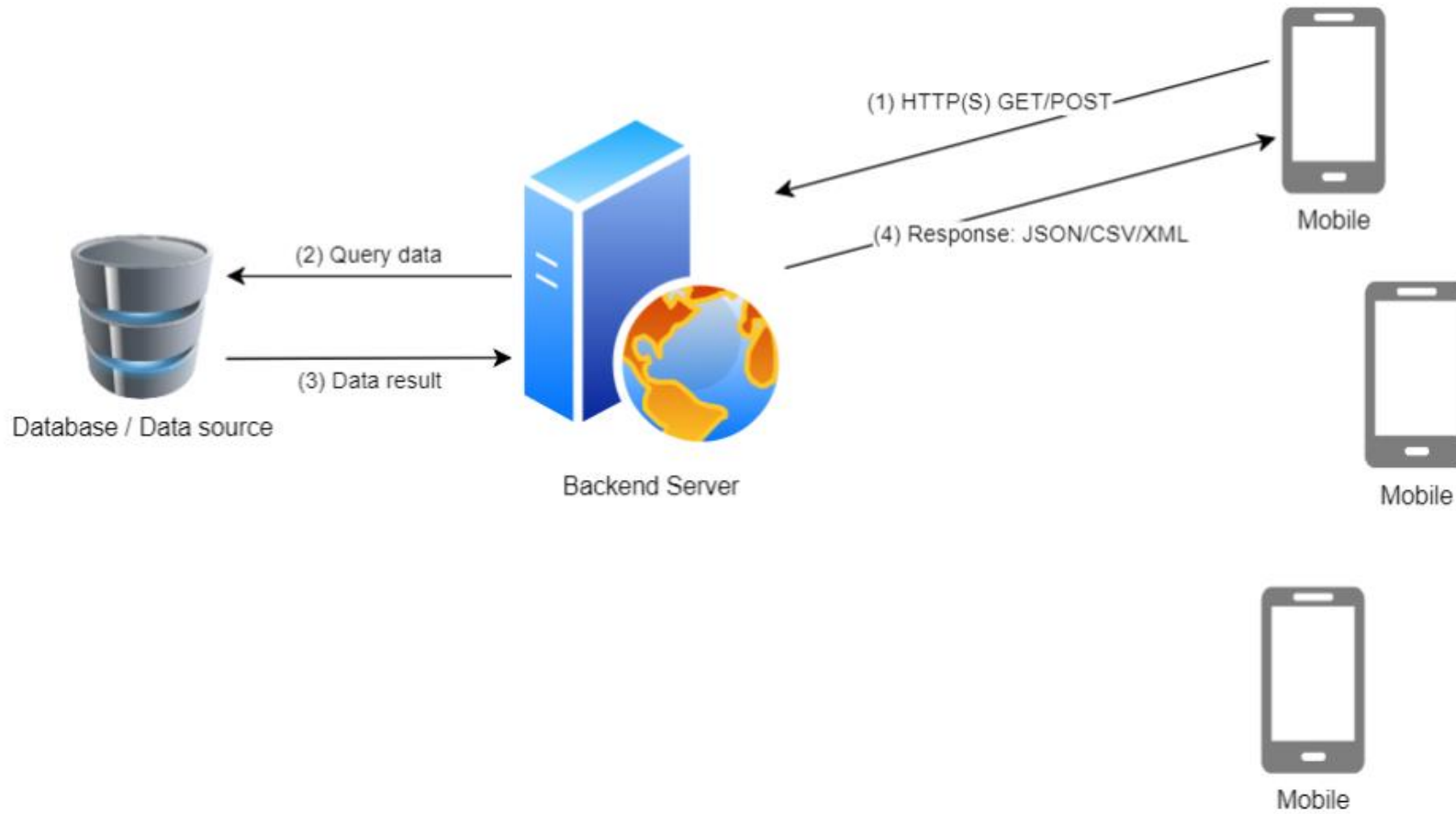
```
val message = ByteArray(1500)
val packet = DatagramPacket(message, message.size)
val socket = DatagramSocket(10100)
socket.receive(packet)
val msg = String(
    message, 0, packet.getLength()
)
Log.d("MYTAG", "received: $msg")
socket.close()
```

TCP/IP és UDP library: Kryonet

- Java könyvtár TCP és UDP kliens-szerver kommunikációhoz
- Kryo sorosító könyvtár
 - > Objektumok küldése hálózaton keresztül
- Mobil és asztali környezet támogatása
- Hatékony
- Játékokhoz kiváló
- Szolgáltatás felderítést támogat
- <https://github.com/EsotericSoftware/kryonet>

HTTP(s) kommunikáció

Tipikus architektúra



HTTP kommunikáció Android platformon

- Egyik leggyakrabban használt kommunikációs technológia
- HTTP metódusok
 - > GET, POST, PUT, DELETE
- Teljes körű HTTPS támogatás és certificate import lehetőség
- REST kommunikáció támogatása (Representational State Transfer)

HTTP kapcsolatok kezelése

- Szükséges engedély:
 - > `<uses-permission android:name="android.permission.INTERNET" />`
- **Új szálban kell megvalósítani a hálózati kommunikáció hívást!**
- Ellenőrizzük a HTTP válasz kódot:
 - > <https://restfulapi.net/http-status-codes/>
 - > <http://www.w3.org/Protocols/HTTP/HTRESP.html>
- HTTP REST
 - > http://en.wikipedia.org/wiki/Representational_state_transfer
- **Ügyeljünk az alapos hibakezelésre**
- HTTP GET példa:
 - > <http://numbersapi.com/3/math>

HTTP könyvtárak Android-on

- A rendszer két megvalósítás is tartalmaz:
 - > Standard Java HTTP implementáció (*HttpURLConnection*)
 - > Apache HTTP implementáció (*HttpClient*)
- Apache Deprecated – Ne használjuk, ki is vették
- Igazán egyik sem jó
 - > 3rd party megoldás – Square OkHttp
 - > <http://square.github.io/okhttp/>

Példa API:

- > Host: <http://api.openweathermap.org/>
- > Path: [data/2.5/weather](#)
- > Params: [?](#)
[q=Budapest](#)
[&](#)
[units=metric](#)
[&](#)
[appid=f3d694bc3e1d44c1ed5a97bd1120e8fe](#)

HTTP GET - HttpURLConnection

```
fun httpGet(urlAddr: String) {
    var reader: BufferedReader? = null
    try {
        val url = URL("http://mysrver.com/api/getitems")
        val conn = url.openConnection() as HttpURLConnection
        reader = BufferedReader(InputStreamReader(conn.inputStream))
        var line: String?
        do {
            line = reader.readLine()
            System.out.println(line)
        } while (line != null)
    } catch (e: IOException) {
        e.printStackTrace()
    } finally {
        if (reader != null) {
            try {
                reader.close()
            } catch (e: IOException) {
                e.printStackTrace()
            }
        }
    }
}
```

HTTP POST- HttpURLConnection

```
fun httpPost(urlAddr: String, content: ByteArray) {  
    // ...  
    var os: OutputStream? = null  
    try {  
        val url = URL("http://mysrver.com/api/refreshitems")  
        val conn = url.openConnection() as HttpURLConnection  
        conn.requestMethod = "POST"  
        conn.doOutput = true  
        conn.useCaches = false  
        os = conn.outputStream  
        os.write(content)  
        os.flush()  
        // ...  
    } catch (e: IOException) {  
        e.printStackTrace()  
    } finally {  
        // ...  
        if (os != null) {  
            try {  
                os.close()  
            } catch (e: IOException) {  
                e.printStackTrace()  
            }  
        }  
    }  
}
```

Timeout értékek beállítása

- Fontos, hogy minden hálózati kommunikáció megfelelő módon kezelje a timeout-ot
- Timeout a kapcsolat megnyitásra
- Timeout az eredmény kiolvasására
- Példa:

...

```
val conn = url.openConnection() as HttpURLConnection
```

...

```
conn.setConnectTimeout(10000)
```

```
conn.setReadTimeout(10000)
```

...

Aszinkron kommunikáció

UI módosítása más szálból

- Az alkalmazás indításakor a rendszer létrehoz egy úgynevezett *main* szálat (UI szál)
- Sokáig tartó műveletek blokkolhatják a felhasználói felületet, ezért új szálbba kell indítani őket
- Az ilyen műveletek a végén az eredményt a UI-on jelenítik meg, **azonban** az Android a UI-t csak a fő szálból engedi módosítani!
- Több megoldás is szóba jöhet:
 - > *Activity.runOnUiThread(Runnable)*
 - > *View.post(Runnable)*
 - > *View.postDelayed(Runnable, long)*
 - > *Handler*
 - > *AsyncTask* és *LocalBroadcast* - *deprecated*
 - > Külső libek, pl. *EventBus* v. *Otto*
 - > *Retrofit*
 - > *Coroutine* - *Kotlin*

Tipikus adatformátumok

Tipikus üzenet formátumok

- CSV, TSV
- XML
 - > Beépített API/parser
 - > SimpleXML külső könyvtár
- JSON:
 - > Beépített API/parser
 - > GSON külső könyvtár
 - > Moshi külső könyvtár
- REST API tesztelésére:
 - > Chrome alkalmazás: PostMan

JSON formátum

- Szintaktikai elemek: '{', '}', '[', ']', ':', ','

- Példa:

```
{
  "keresztnev" : "Elek",
  "vezeteknev" : "Teszt",
  "kor" : 23,
  "cim" :
  {
    "utca" : "Baross tér",
    "varos" : "Budapest",
    "iranyitoszam" : "1087"
  },
  "telefon":
  [
    {
      "tipus" : "otthoni",
      "szam": "123 322 1234"
    },
    {
      "tipus" : "mobil",
      "szam": "626 515 1567"
    }
  ]
}
```

JSON feldolgozás

- *JSONObject*:
 - > JSON objektumok parse-olása
 - > Elemek elérhetősége a kulcs megadásával:
 - `getString(String name)`
 - `getJSONObject(String name)`
 - `getJSONArray(String name)`
 - > JSON objektum létrehozása *String*-ből vagy *Map*-ból
- *JSONArray*:
 - > *JSONObject*-hez hasonló működés JSON tömbökkel
 - > Parse-olás, elemek lekérdezése index alapján, hossz
 - > Létrehozás például *Collection*-ből

JSON -> Kotlin osztály

```
{  
  "base": "USD",  
  "rates": {  
    ...  
    "HUF": 289.8969072165,  
    "EUR": 0.896458987  
  },  
  "date": "2019-05-04"  
}
```

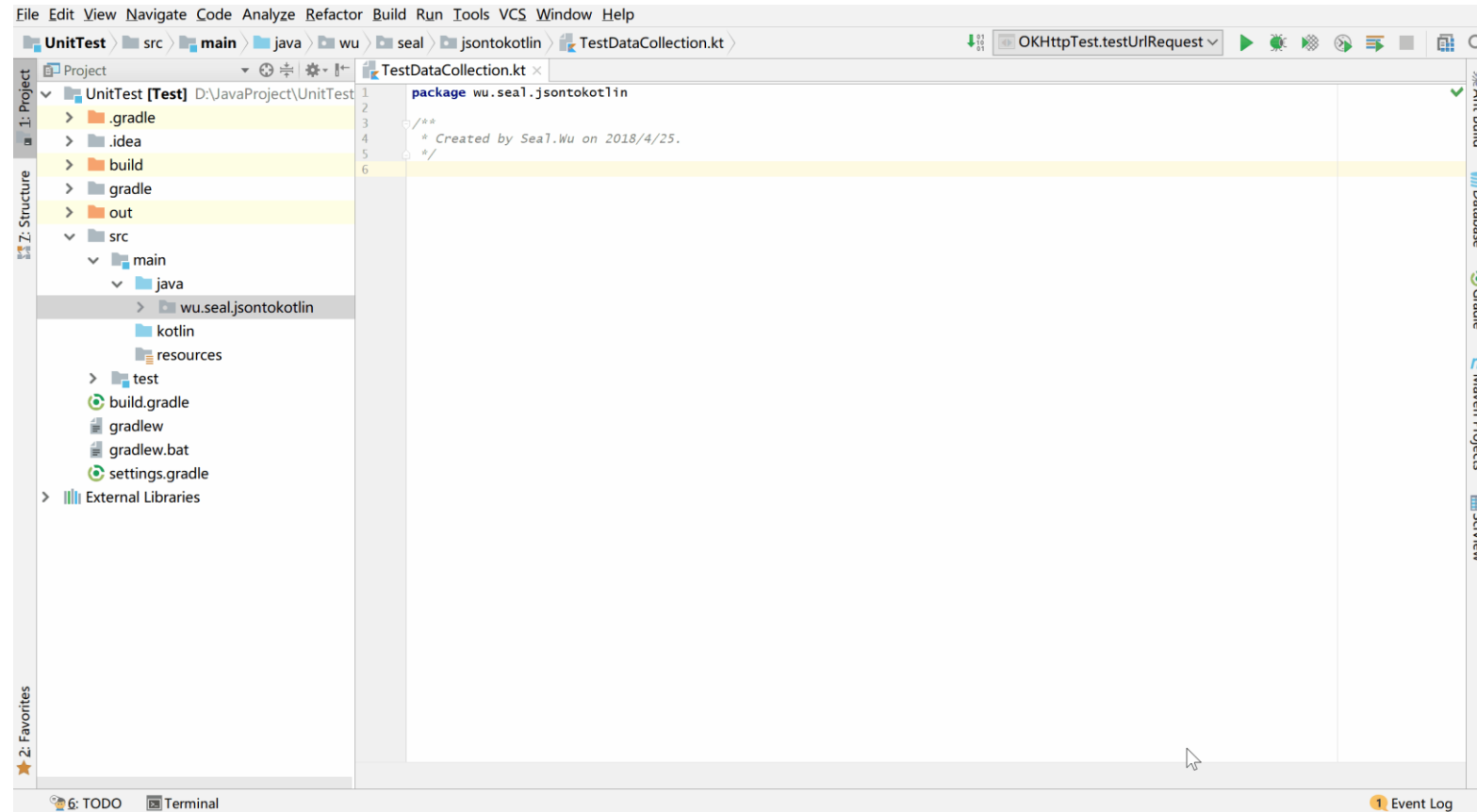


```
data class MoneyResult(  
    val date: String?,  
    val rates: Rates?,  
    val base: String?  
)  
  
data class Rates(  
    ...  
    val HUF: Number?,  
    val EUR: Number?  
)
```

Entitás vagy *data* class generálás JSON-ból

- JSON to Kotlin Android Studio Plugin

- > <https://plugins.jetbrains.com/plugin/9960-json-to-kotlin-class-jsontokotlinclass->



JSON API minták

- Currency Exchange:
 - > <https://www.freeforexapi.com/api/live?pairs=USDHUF>
 - > <https://cdn.jsdelivr.net/gh/fawazahmed0/currency-api@1/latest/currencies/usd.json>
 - > http://data.fixer.io/api/latest?access_key=969c37b5a73f8cb2d12c081dcbdc35e6
- Stocks:
 - > https://polygon.io/docs/stocks/get_v2_aggs_ticker__stocksticker__range__multiplier__timespan__from__to
- OpenWeather
 - > <http://api.openweathermap.org/data/2.5/weather?q=Budapest,hu&units=metric&appid=f3d694bc3e1d44c1ed5a97bd1120e8fe>
- TV Show Data API:
 - > <http://api.tvmaze.com/search/shows?q=stargate>
- NASA APIs:
 - > <https://api.nasa.gov/>
- JSON olvasható formátumban:
 - > <https://jsonformatter.curiousconcept.com>

XML formátum

```
<?xml version="1.0"?>  
<employees>  
  <person>  
    <name>Big Joe</name>  
    <address>Beach Street 12.</address>  
    <phone>111-222</phone>  
  </person>  
  <person>  
    <name>Small Joe</name>  
    <address>Hill Street 13.</address>  
    <phone>222-333</phone>  
  </person>  
</employees>
```


XML feldolgozás

- Az Android gazdag eszközkészletet biztosít XML-ek feldolgozására
- SAX alapú feldolgozás
 - > *javax.xml.parsers.SAXParser*
 - > Különbféle függvényekkel dolgozhatjuk fel az értelmező által generált eseményeket
 - > Az eseményeket akkor generálja az értelmező, amikor a jelölő nyelv meghatározott részeihez ér
- DOM alapú feldolgozás
 - > *javax.xml.parsers.DocumentBuilder*
 - > *javax.xml.parsers.DocumentBuilderFactory*
 - > Memóriába kerül beolvasásra az XML mint egy „fa”
 - > Lekérdezhetők az elemek

Külső osztálykönyvtárak XML és JSON feldolgozásra

- XML:
 - > SimpleXML
- JSON:
 - > GSON
 - > Moshi
 - > <http://www.jsonschema2pojo.org/>
- REST API tesztelésére:
 - > PostMan (Chrome plugin)

REST libraryk

- OkHttp
- Volley
- RESTLet
- Retrofit
 - > OkHttp-re épül

Retrofit

Retrofit

- HTTP kérések leírása annotációkkal:
 - > URL és query paraméterek
 - > Body – objektum konverzió (JSON, protocol buffers)
 - > Multipart request és file feltöltés
- Gradle:
 - > implementation 'com.squareup.retrofit2:retrofit:2.8.1'
- További információk:
 - > <http://square.github.io/retrofit/>

Retrofit – KotlinX Serialization támogatás

- Automatikus konverzió a háttérben

- > Be kell állítani a Retrofitnek hogy mit használjon a konverzióhoz.

```
val retrofit=Retrofit.Builder()  
    .baseUrl("https://api.exchangeratesapi.io/")  
    .addConverterFactory(Json.asConverterFactory(  
        "application/json".toMediaType()))  
    .build()
```

További információk:

- > <http://square.github.io/retrofit/>

Retrofit -Gradle

```
plugins {  
    id("com.google.devtools.ksp") version "1.9.23-1.0.20" apply false  
    id("org.jetbrains.kotlin.plugin.serialization") version "1.9.24" apply  
false  
}  
...
```

```
// Retrofit
```

```
implementation("com.squareup.retrofit2:retrofit:2.11.0")  
implementation("org.jetbrains.kotlinx:kotlinx-serialization-json:1.6.3")  
implementation("com.jakewharton.retrofit:retrofit2-kotlinx-serialization-  
converter:1.0.0")  
implementation("com.squareup.moshi:moshi:1.15.1")
```

Retrofit - Entitások

```
data class MoneyResult (  
    var date: String,  
    var rates: Rates,  
    var base: String  
)
```

```
data class Rates (  
    var BGN: Double,  
    var CAD: Double,  
    ...  
)
```


Retrofit – API interface

```
import retrofit2.Call
import retrofit2.Callback
import retrofit2.http.GET
import retrofit2.http.Query
```

```
interface CurrencyExchangeAPI {
    @GET ("/latest")
    fun getRates (@Query ("base") base: String) :
    Call<MoneyResult>
}
```

Retrofit - használat

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

    val retrofit = Retrofit.Builder()
        .baseUrl("https://api.exchangeratesapi.io/")
        .addConverterFactory(GsonConverterFactory.create())
        .build()

    val currencyAPI = retrofit.create(CurrencyExchangeAPI::class.java)

    btnGetRate.setOnClickListener {
        val ratesCall = currencyAPI.getRates("EUR")
        ratesCall.enqueue(object: Callback<MoneyResult> {
            override fun onFailure(call: Call<MoneyResult>, t: Throwable) {
                tvResult.text = t.message
            }

            override fun onResponse(call: Call<MoneyResult>,
                response: Response<MoneyResult>) {
                tvResult.text = response.body()?.rates?.HUF.toString()
            }
        })
    }
}
```

Mi nem igaz a kommunikációra Androidon?

- A. Az Android támogatja a Bluetooth LE-t.
- B. A webes eléréshez engedély kell.
- C. A Retrofit automatikusan végzi a JSON objektumokká konvertálását.
- D. A hálózati hívást a UI szálon kell végezni.

BroadcastReceiver

Android komponentens

Broadcast események

- Rendszerszintű eseményekre fel lehet iratkozni – Broadcast üzenet
- Az Intent alkalmas arra hogy leírja az eseményt
- Sok beépített Broadcast Intent, lehet egyedi is

```
ACTION_TIME_TICK  
ACTION_TIME_CHANGED  
ACTION_TIMEZONE_CHANGED  
ACTION_BOOT_COMPLETED  
ACTION_PACKAGE_ADDED  
ACTION_PACKAGE_CHANGED  
ACTION_PACKAGE_REMOVED  
ACTION_PACKAGE_RESTARTED  
ACTION_PACKAGE_DATA_CLEARED
```

```
ACTION_UID_REMOVED  
ACTION_BATTERY_CHANGED  
ACTION_POWER_CONNECTED  
ACTION_POWER_DISCONNECTED  
ACTION_SHUTDOWN
```

Broadcast események

- Nem csak az Android, hanem alkalmazások (Activity-k és Service-ek) is dobhatnak Broadcast Intentet
 - > Telephony service küldi az ACTION_PHONE_STATE_CHANGED Broadcast Intentet, ha a mobilhálózat csatlakozás megváltozott
 - > android.provider.Telephony.SMS_RECEIVED
 - > Sok más, érdemes tájékozódni ha valamit szeretnénk lekezelni
 - > Saját alkalmazásunkból is dobhatunk a `sendBroadcast(String action)` metódussal
 - > Ez is Intent, lehet Extra és Data része

Broadcast intentek elkapása

- Broadcast Receiver nevű komponens segítségével
 - > Kódból vagy manifestben kell regisztrálni
 - (bizonyos Action-ök esetén nem mindegy, tájékozódni!, pl. TIME_TICK)
 - > Intent filterrel állíthatjuk be hogy milyen Intent esetén aktivizálódjon
- Nem Activity, nincs felhasználói felülete
- Azonban képes Activity-t indítani
- Használata: BroadcastReceiver osztályból származtatunk, és felüldefiniáljuk az onReceive() metódust, majd intent-filter

SMS Receiver 1/2

- Maga a Receiver:

```
class SMSReceiver : BroadcastReceiver() {  
    override fun onReceive(context: Context, intent: Intent) {  
        val extras = intent.extras ?: return  
        val pdus = extras.get("pdus") as Array<ByteArray>  
        for (pdu in pdus) {  
            val msg = SmsMessage.createFromPdu(pdu)  
            val origin = msg.originatingAddress  
            val body = msg.messageBody  
            Toast.makeText(context,  
                "SMS caught, number: $origin body: $body", Toast.LENGTH_LONG).show()  
        }  
    }  
}
```


SMS Receiver 2/2

- Regisztráció:

- > Statikusan: AndroidManifest.xml

```
<uses-permission android:name="android.permission.RECEIVE_SMS"/>

...
<receiver android:name=".SMSReceiver" android:enabled="true">
  <intent-filter android:priority="1000">
    <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
  </intent-filter>
</receiver>
```

- > Dinamikusan:

```
val smsReceiver = SMSReceiver
override fun onStart() {
    super.onStart()
    LocalBroadcastManager.getInstance(requireContext())
        .registerReceiver(smsReceiver, IntentFilter(android.provider.Telephony.SMS_RECEIVED))
}
override fun onStop() {
    LocalBroadcastManager.getInstance(requireContext())
        .unregisterReceiver(smsReceiver)
    super.onStop()
}
```

Broadcast intentek kezelése

- Broadcast továbbdobásának megakadályozása:
 - > `abortBroadcast();`
- Például ha a fülhallgató média gombjait kell kezelni és nem akarjuk, hogy a zenelejátszó is megkapja a Broadcast-ot 😊

Alkalmazáskomponens indítása Boot után

- Néha olyan szolgáltatásokra van szükség, amelyek mindig futnak a készüléken
- Ilyen esetben fontos, hogy a készülék indítása esetén ezek automatikusan is el tudjanak indulni
- Az Android lehetőséget biztosít arra, hogy feliratkozzunk a „*Boot befejeződött*” eseményre és valamilyen alkalmazás komponensst elindítsunk:
 - > *BroadcastReceiver* definiálása Manifest-ben
 - > *android.intent.action.BOOT_COMPLETED*
- A *BroadcastReceiver onReceive()* függvényében elindíthatjuk a megfelelő komponensst

Service

Android komponens

Bevezetés

- Összetett alkalmazások esetén gyakran szükség van **háttérben futó folyamatokra**, amelyek **felhasználói felület nélküli** funkcionalitást valósítanak meg (pl. hálózati kommunikáció, monitorozás, zene lejátszás, fájl feltöltés, stb...)
- Android alkalmazáskomponens: **Service**
- Többféle Service típus és viselkedési modell
- Ügyeljünk a megfelelő leállításra és az erőforrás felszabadításra!
- Minden Service létrehozás komoly felelősség a fejlesztő részéről!

Service bemutatása

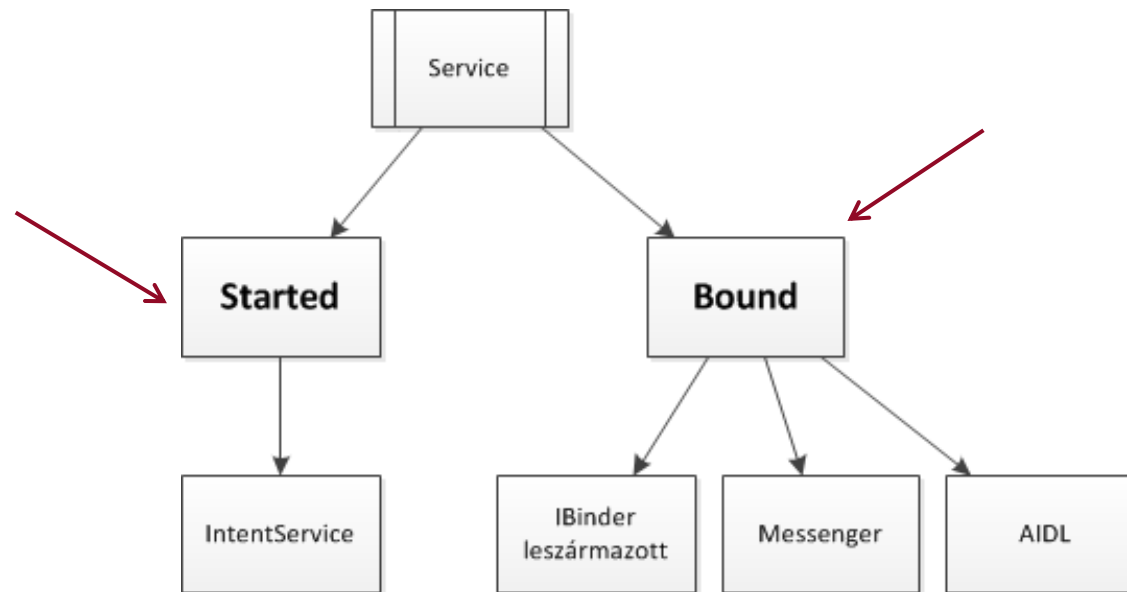
- Valamilyen hosszabb ideig tartó, háttérben futó feladatot lát el
- Felhasználói beavatkozás nélkül működik
- Nincs felhasználói felülete
- Más komponensek/alkalmazások számára is szolgáltathat funkciókat
- Bármilyen alkalmazáskomponens elindíthatja és **akkor is futva maradhat, amikor a hívó** (pl. Activity) **megáll**, tipikusan ha más alkalmazásra váltunk

Service bemutatása

- Más alkalmazások komponensei is kapcsolódhatnak hozzá és kommunikálhatnak vele (*Interprocess Communication*, IPC)
- (ahogy egy Activity is elindítható másik alkalmazásból, de a **Service**-nél ez letiltható)
- Ugyanúgy **Intent**-el indítjuk el
- Példa szolgáltatások:
 - > Zenelejátszó
 - > Hosszabb hálózati kommunikáció (pl. torrent)
 - > GPS pozíció követés - rendszerszolgáltatás
 - > Stb...

Service típusok

- Egy Service kétféle módon képes működni:
 - > *Komponensből indított / **Started***
 - > *Kapcsolt / **Bound***



Background Service limitációk

- Android 8-tól felfele
 - > De korábbi verziókban is beállítható manuálisan
- Limitált végrehajtás, sokkal nagyobb valószínűséggel leállítja a rendszer
- Néhány perc után idle-be állítja az alkalmazást, ami olyan mintha a `stopSelf` meghívódott volna()

Service működési típusok

1. Komponensből indított (***Started/Unbound***):
 - > Valamilyen alkalmazás komponens (tipikusan Activity) elindítja a ***startService(intent)*** metódussal
 - > A Service akkor is folytathatja a futását, amikor a hívó komponens már megsemmisült
 - A hívó megállítása nem törli a Service-t is!
 - > Általában az így indított Service-ek **egy feladatot hajtanak végre**, nem folytonos szolgáltatást nyújtanak (például egy darab fájl feltöltése vagy letöltése)
 - > Csak a hívónak van rá referenciája
 - > Ha végzett, **magát kell leállítania** a *stopSelf()* metódussal vagy a hívónak *stopService(intent)*-el, az op.rendszer nem fogja!

Service működési típusok

2. Kapcsolt szolgáltatás (*Bound*):

- > Nem indítjuk „kézzel”, **magától indul** ha kapcsolódni próbálunk hozzá
- > Addig fut amíg van hozzá kapcsolódó komponens
- > **Egyszerre többen is kapcsolódhatnak**, akár más alkalmazásból is
- > **Hosszabb, folyamatosan tartó feladatokhoz**
 - Pl. Torrent, zenelejátszó
- > **Nem kell önmagát leállítania**, az Android gondoskodik róla
- > Gyakorlatilag minden **rendszer szolgáltatás** ilyen

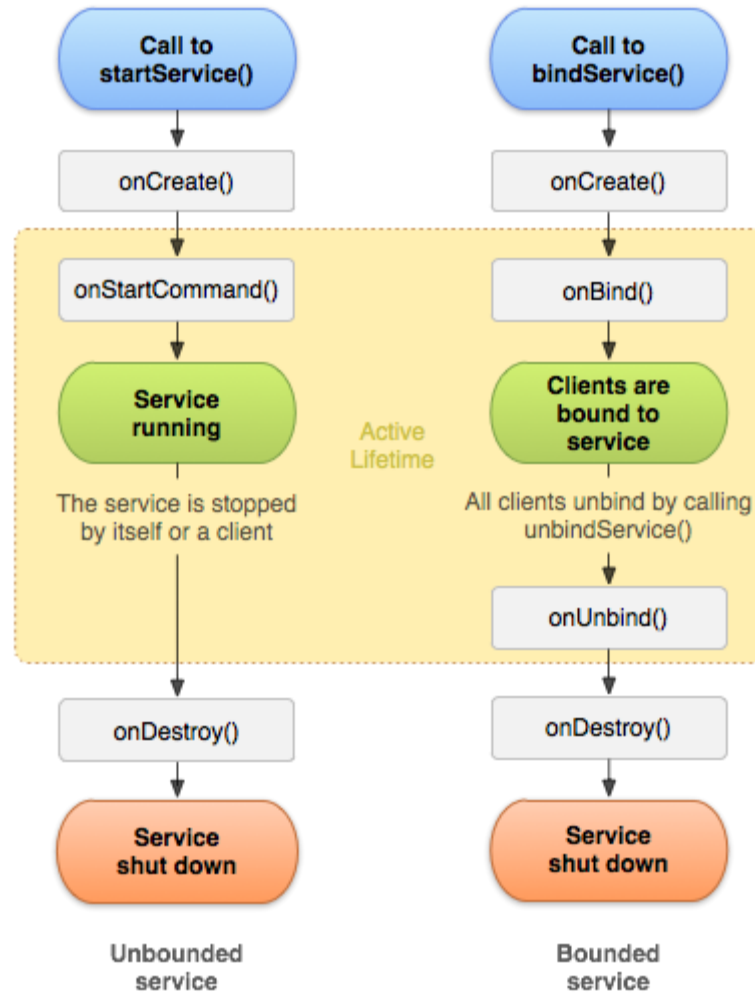
Service működési típusok

- A Service támogathatja **egyszerre mind a két típusú működést** is (!), néha szükséges is
- Started esetben nem tudunk referenciát szereznni a Service-re, ha a hívó (Activity) már nem fut
- Példa: zenelejátszó ami mindkét módot igényli
 1. A lejátszó UI-ról indítjuk a playlist lejátszást `startService()`-el
 2. Kilépünk az alkalmazásból, a Service fut a háttérben és megy a zene
 3. Visszatérünk a lejátszóba hogy számot váltsunk. Ekkor az új Activity példányból már nem tudunk referencát szereznni a Started Service-re, és megkérni hogy váltszon a következő számra
 4. Tudunk viszont kapcsolódni a Service-hez, ha implementáltuk a Bound működési módot is
 5. Kapcsolódunk a Service-hez, és utasítjuk a számváltásra

Service megvalósítás

- Service (vagy beépített gyerek) osztályból származtatunk
- Megvalósítjuk a megfelelő callback függvényeket:
 - > Ha *Started*-ként (is) akarjuk használni: ***onStartCommand()*** implementálása
 - > Ha *Bound*-ként (is) akarjuk használni: ***onBind()*** implementálása

Service életciklus modell



Forrás: <https://developer.android.com/guide/components/services>

Service kezelés

- Mindkét működés esetén más alkalmazás is használhatja a szolgáltatást
- Ugyanúgy, mint az Activity-t is – **Intentek** segítségével, akár más alkalmazásból is
- De! Lehetőség van **privát szolgáltatások** létrehozására is:
 - > Manifest attribútumban megadva (ld. később)
 - > Ekkor más alkalmazásból nem férnek hozzá a classname megadásával sem

Műveletek végrehajtása Service-n belül

- A Service alapértelmezetten a processze fő szálában fut, **nem kap külön szálát!!**
- Erőforrás igényes műveletek esetén „kézzel” kell új szálát indítanunk, pl.:
 - > CPU intenzív feladatok (pl. titkosítás, enkódolás, zene lejátszás)
 - > Hálózati kommunikáció, stb...
- Ellenkező esetben 5mp után ugyanúgy megjelenik az *Application Not Responding* (ANR) ablak, mint Activity esetén

Service készítése

1. Leszármaztatunk az *android.app.Service*-ből

```
import android.app.Service
```

```
class DemoService : Service() {  
    ...  
}
```

2. Szükséges callback függvények megvalósítása

```
override fun onCreate() {  
    super.onCreate()  
}
```

```
override fun onBind(intent: Intent): IBinder? {  
    return null  
}
```

```
override fun onStartCommand(intent: Intent, flags: Int, startId: Int): Int {  
    //to things  
}
```

3. Regisztráció a Manifestben

Futási idejű Engedélyek (Runtime permissions)

Mire szolgálnak az engedélyek?

- Veszélyes/kritikus/személyes adatokat érintő műveletekhez engedélyre van szükség a felhasználótól.
- Engedély kérés régebben:
 - > Manifest állományban egyszerű bejegyzés:
`<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>`
 - > Majd az összes engedély egyszeri elkérése telepítéskor
- Új Permission modell Android 6 óta:
 - > Veszélyes engedélyeket futási időben kell kérni

Futási idejű permission kezelés

- Permission meglétének ellenőrzése:

```
val permissionCheck =  
    ContextCompat.checkSelfPermission(  
        thisActivity,  
        Manifest.permission.WRITE_CALENDAR  
    )
```

- Permission kérés Activity-ben:
 - > Megvizsgálni, hogy megvan-e már az engedély
 - > Felhasználó tájékoztatása az engedély kérés okáról
 - > Permission kérés eredményének kezelése egyesével
 - Engedélyezés
 - Tiltás

Engedély típusok

- Manifest állományban továbbra is érdemes felsorolni a permission-öket
 - > Normál permission-öket automatikusan megkap az alkalmazás
 - > Veszélyes permission-öket kérni kell kódból
- Normál és veszélyes permission-ök listája:
 - > <https://developer.android.com/guide/topics/security/permissions.html#normal-dangerous>
- Korábban megadott engedélyeket a felhasználó a beállításokban visszavonhatja!
- További részletek:
 - > <https://developer.android.com/training/permissions/requesting.html>

Permission kérés 1/2

- Manifest file-ba regisztráció
- Engedély kérés kódból:

```
fun requestNeededPermission() {  
    if (ContextCompat.checkSelfPermission(this,  
        android.Manifest.permission.CAMERA) !=  
        PackageManager.PERMISSION_GRANTED) {  
        if (ActivityCompat.shouldShowRequestPermissionRationale(this,  
            android.Manifest.permission.CAMERA)) {  
            Toast.makeText(this,  
                "I need it for camera", Toast.LENGTH_SHORT).show()  
        }  
  
        ActivityCompat.requestPermissions(this,  
            arrayOf(android.Manifest.permission.CAMERA),  
                PERMISSION_REQUEST_CODE)  
    } else {  
        // már van engedély  
    }  
}
```

Ha először lett
utasítva,
megjeleníthetünk
magyarázatot

Permission kérés 2/2

```
override fun onRequestPermissionsResult (  
    requestCode: Int,  
    permissions: Array<String>,  
    grantResults: IntArray  
) {  
    when (requestCode) {  
        PERMISSION_REQUEST_CODE -> {  
            if (grantResults.isNotEmpty() && grantResults[0] ==  
                PackageManager.PERMISSION_GRANTED) {  
                Toast.makeText(this, "CAMERA perm granted",  
                    Toast.LENGTH_SHORT).show()  
            } else {  
                Toast.makeText(this, "CAMERA perm NOT granted",  
                    Toast.LENGTH_SHORT).show()  
            }  
        }  
    }  
}
```

Futási idejű engedélyek

- Külső osztálykönyvtárakkal is lehet kezelni
- **Permission Dispatcher**
 - > <https://github.com/permissions-dispatcher/PermissionsDispatcher>
- Dexter
 - > <https://github.com/Karumi/Dexter>
- Android Runtime Permission library
 - > <https://github.com/nabinbhandari/Android-Permissions>
- Mayl
 - > <https://github.com/ThanosFisherman/Mayl>
- Egyebek
 - > <https://android-arsenal.com/tag/235>

Legjobb gyakorlatok permission kezelésre

- Használjuk lehetőleg a rendszer beépített szolgáltatásait (pl. kamera)!
- Csak a szükséges engedélyeket kérjük el!
- Ne terheljük túl a felhasználót kérésekkel!
 - > Csak akkor kérjük el az engedélyt, amikor szükség van rá!
- Magyarázzuk el a felhasználónak, hogy miért van szükség az adott engedélyre!
- Teszteljük mindkét permission kezelést (manifest és Kotlin kód) korábbi és új Android verziókon!

Engedélyek kezelése Compose esetén

1. Manuális kezelése, Compose és Activity együttműködés

> *rememberLauncherForActivityResult*

2. Accompanist permissions könyvtár használata (by Google)

> `Implementation("com.google.accompanist:accompanist-permissions:0.5.0-alpha")`

```
val cameraPermissionState = rememberPermissionState(
    android.Manifest.permission.CAMERA
)
val cameraLauncher = rememberLauncherForActivityResult(
    contract = ActivityResultContracts.TakePicture(),
    onResult = { success ->
        hasImage = success
    }
)
```

- További részletek:

> <https://betterprogramming.pub/jetpack-compose-request-permissions-in-two-ways-fd81c4a702c>

Engedélyek kezelése Compose esetén

```
if (cameraPermissionState.status.isGranted) {  
    ...  
} else {  
    Column {  
        val textToShow = if (cameraPermissionState.status.shouldShowRationale) {  
            // If the user has denied the permission but the rationale can be shown,  
            // then gently explain why the app requires this permission  
            "The camera is important for this app. Please grant the permission."  
        } else {  
            // If it's the first time the user lands on this feature, or the user  
            // doesn't want to be asked again for this permission, explain that the  
            // permission is required  
            "Camera permission required for this feature to be available. " +  
            "Please grant the permission"  
        }  
        Text(textToShow)  
        Button(onClick = { cameraPermissionState.launchPermissionRequest() }) {  
            Text("Request permission")  
        }  
    }  
}
```

Hogy is volt?

- Milyen rövidtávú kommunikációs technológiákat ismer?
- Milyen hosszútávú kommunikációs technológiákat ismer?
- Mi a különbség az UDP és a TCP/IP alapú protokollok között?
- Hogyan kell végezni a hálózati kommunikációt Androidon?
- Mire szolgál a Retrofit könyvtár?
- Milyen módszereket ismer a BroadcastReceiverek regisztrálására?
- Hogyan biztosítható, hogy egy bizonyos típusú broadcastot megkapjon a komponensünk?
- Milyen típusú Service-eket ismer?
- Mire kell nagyon figyelni Service írásakor?
- Milyen engedély kategóriákat ismer?
- Milyen fázisai vannak a veszélyes engedélyek elkérésének?

Összefoglalás

- Rövidtávú kommunikáció:
 - > NFC
 - > Bluetooth
- Internet alapú kommunikáció:
 - > TCP/IP, UDP
 - > HTTP(s) kapcsolatok kezelése
 - Retrofit
 - > Tipikus adatformátumok
- Broadcast Receiver
- Service
- Futási idejű engedélyek

Köszönöm a figyelmet!

