

# Firestore Backend as a Service, kamera kezelés

Ekler Péter

BME VIK AUT, AutSoft

[peter.ekler@aut.bme.hu](mailto:peter.ekler@aut.bme.hu)



# Tematika

1. Android platform bemutatása, Kotlin alapok
2. Alkalmazás komponensek, Kotlin konvenciók
3. Felhasználói felület
4. Fragmentek, haladó UI
5. Listák kezelése hatékonyan
6. Perzisztens adattárolás, adatbázisok, haladó Kotlin
7. Felhő szolgáltatások
8. Hálózati kommunikáció
9. Helymeghatározás, térkép kezelés
10. Architektúra komponensek, JetPack

# Tartalom

- Perzisztens adattárolás
  - > SQLite
  - > ORM - Room
  - > SharedPreferences
  - > File kezelés
- Adattárolás a felhőben
- Firebase Backend as a Service
  - > Felhasználók kezelése
  - > Real-time adatkezelés
  - > Képek feltöltése
  - > Crash reporting
  - > Push notification

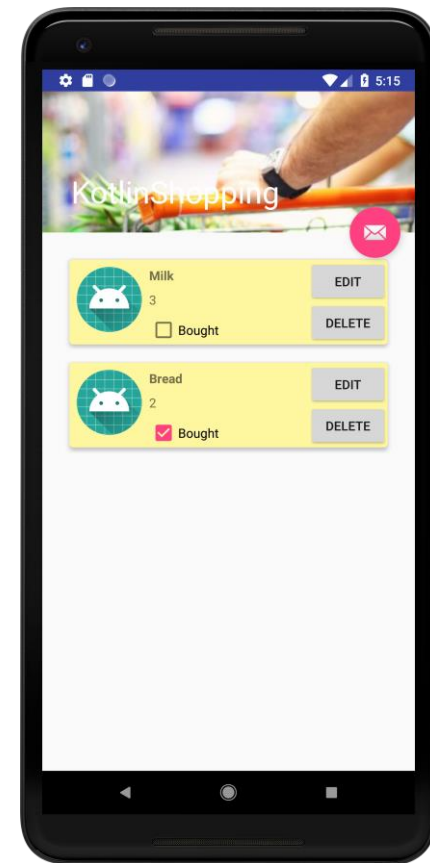
# Házi feladat

- Bevásárló lista állapot mentés

```
holder.cbBought.setOnClickListener{  
    items[position].bought = holder.cbBought.isChecked  
    val dbThread = Thread {  
        AppDatabase.getInstance(context).shoppingItemDao()  
            .updateItem(items[position])  
    }  
    dbThread.start()  
}
```

- Törlés

```
private fun deleteItemBasedOnPosition(position: Int) {  
    val dbThread = Thread {  
        AppDatabase.getInstance(context).shoppingItemDao()  
            .deleteItem(items[position])  
    }  
    dbThread.start()  
    items.removeAt(position)  
    notifyItemRemoved(position)  
}
```



# SharedPreferences

Beállítások mentése hosszú távra

# SharedPreferences

- Alaptípusok tárolása kulcs-érték párokként (~*Dictionary*)
  - > Típusok: *int*, *long*, *float*, *String*, *boolean*
- Fájlban tárolódik, de ezt elfedi az operációs rendszer
- Létrehozáskor beállítható a láthatósága
  - > **MODE\_PRIVATE**: csak a saját alkalmazásunk érheti el
  - > **MODE\_WORLD\_READABLE**: csak a saját alkalmazásunk írhatja, bárki olvashatja
  - > **MODE\_WORLD\_WRITABLE**: bárki írhatja és olvashatja
- Megőrzi tartalmát az alkalmazás és a telefon újraindítása esetén is
  - > Miért?

# SharedPreferences

- Ideális olyan adatok tárolására, melyek primitív típussal könnyen reprezentálhatók, pl:
  - > Default beállítások értékei
  - > UI állapot
  - > Settings-ben megjelenő adatok (innen kapta a nevét)
- Több ilyen *SharedPreferences* fájl tartozhat egy alkalmazáshoz, a nevük különbözteti meg őket
  - > **getSharedPreferences(name: String, mode: Int)**
  - > Ha még nem létezik ilyen nevű, akkor az Android létrehozza
- Ha elég egy SP egy Activity-hez, akkor nem kötelező elnevezni
  - > **getPreferences(mode: Int)**

# SharedPreferences írás

- Közvetlenül nem írható, csak egy *Editor* objektumon keresztül

```
val PREF_NAME: String = "MySettings"
val sp: SharedPreferences =
    getSharedPreferences(PREF_NAME, MODE_PRIVATE)
    editor: Editor = sp.edit()
    editor.putLong("lastSyncTimestamp",
        Calendar.getInstance().getTimeInMillis())
    editor.putBoolean("KEY_FIRST", false)
    editor.apply()
```

Azonosító (fájlnév)

Csak mi érjük el

Érték  
típusa

Megnyitjuk írásra

Kulcs

Érték

Változtatások  
mentése (kötelező!!!)



# SharedPreferences olvasás

- Az *Editor* osztály nélkül olvasható, közvetlenül a SharedPreferences objektumból
- Ismernünk kell a kulcsok neveit és az értékek típusát
  - > Emiatt sem alkalmas nagy mennyiségű adat tárolására

```
PREF_NAME = "MySettings"
val sp =
    getSharedPreferences(PREF_NAME, MODE_PRIVATE)
val lastSaved: Long = sp.getLong("lastSaved", 0)
val isFirstRun: Boolean =
    sp.getBoolean("KEY_FIRST", true)
```

Tudni kell a kulcsot

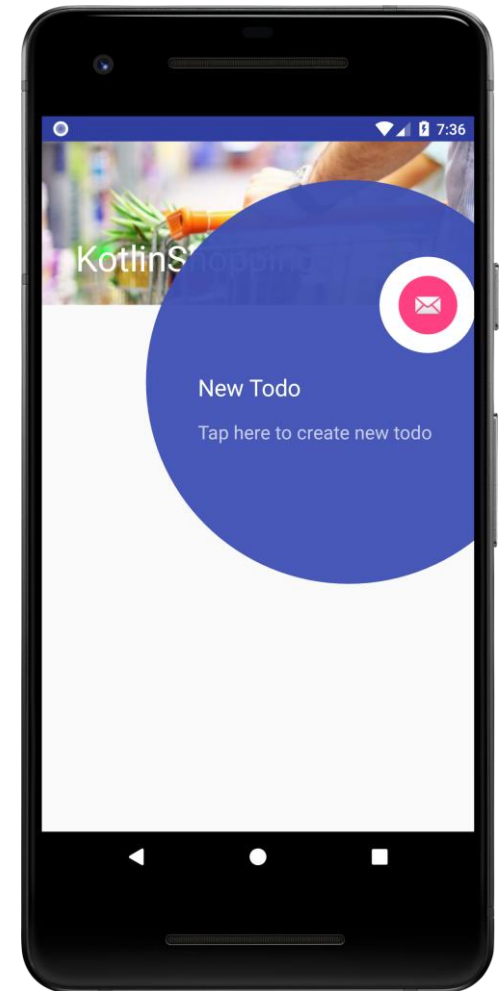
És a típust is!

Alapértelmezett  
érték

- Egy hasznos metódus:
  - > **sp.getAll()** – minden kulcs-érték pár egy Map objektumban
  - > Tutorial lib: <https://github.com/sjwall/MaterialTapTargetPrompt>

# Gyakoroljunk!

Egészítsük ki a bevásárló lista alkalmazást, hogy csak legelső induláskor mutasson használati tippeket.



# Fájlkezelés - Internal storage

# Internal storage

- Alkalmazás saját, védett lemezterülete
- `mnt/sdcard/data/data/[Package name]` könyvtár
- Fájl írása:

```
val FILENAME = "hello_file.txt"
val content = "Hello World!"
val out: FileOutputStream =
    openFileOutput(FILENAME, Context.MODE_PRIVATE)
out.write(content.getBytes())
out.close()
```

Internal Storage  
könyvtárba ír



- `openFileOutput()`-tal csak a könyvtár gyökerébe tudunk fájlokat írni

# Internal storage

- `openFileOutput(filename: String, mode: Int)`
  - > filename-ben nem lehet „\”, egyébként kivételt dob (Miért?)
  - > Támogatott módok:
    - `Context.MODE_PRIVATE`: alapértelmezett megnyitási mód, felülírja a fájlt ha már van benne valami
    - `Context.MODE_APPEND`: hozzáfűzi a fájlhoz amit beleírunk
    - Lehet `WORLD_READABLE` vagy `WORLD_WRITEABLE` is, ha szükséges, de nem ez a javasolt módja az adatok kiajánlásának, hanem a `ContentProvider` (később)
  - > *Privát vagy Append* mód esetén nincs értelme kiterjesztést megadni, mert máshonnan úgysem fogják megnyitni
  - > Ha nem létezik a fájl akkor létrehozza, a **WORLD\_\*** módok csak ekkor értelmezettek

# Internal storage

- Fájl olvasása ugyanígy:
  - > **openFileInput(filename: String)** hívása (FileNotFoundException-t dobhat)
  - > Byte-ok kiolvasása a visszakapott *FileInputStream*-ből a **read()** metódussal
  - > Stream bezárása *close()* metódussal!
- Cache használata
  - > Beépített mechanizmus arra az esetre, ha cache-ként akarunk fájlokat használni
  - > **getCacheDir()** metódus visszaad egy File objektumot, ami a cache könyvtárra mutat (miért File?)
  - > Ezen belül létrehozhatunk cache fájlokat
  - > Kevés lemezterület esetén először ezeket törli az Android
    - Nem számíthatunk rá, hogy mindig ott lesznek!
  - > Google ajánlás: maximum 1MB-os fájlokat rakjunk ide (Miért?)

# Statikus fájlok egy alkalmazáshoz

- Szükséges lehet a fejlesztett alkalmazáshoz statikusan fájlokat linkelni
  - > Kezdeti, nagy méretű, feltöltött bináris adatbázis fájl
  - > Egyedi formátumú állomány
  - > Bármilyen fájl, de nem illik a res könyvtár mappáiba (drawable, xml, stb)
- Fejlesztéskor a **res/raw** mappába kell raknunk őket
- Ezek telepítéskor szintén az internal storage-be kerülnek
- Read-only lesz telepítés után, nem tudjuk utólag módosítani
- Olvasásuk futásidőben:

```
val inStream: InputStream =  
    resources.openRawResource(R.raw.myfile)
```

# Statikus fájlok egy alkalmazáshoz

- Mivel ugyanolyan resource mint az összes többi, különböző fájlok használhatók különböző konfigurációkhoz, mint például a UI finomhangolásnál:
  - > **res/layout**: felhasználói felületek alapértelmezett orientáció esetén (telefon: álló, tablet és Google TV: fekvő)
  - > **res/layout-port**: felhasználói felületek álló orientáció esetén
  - > **res/layout-land**: felhasználói felületek fekvő orientáció esetén
- Ugyanúgy lehet ezt is finomhangolni:
  - > **res/raw/initialDatabase.db**: kezdeti adatbázis
  - > **res/raw-hu\_rHU/initialDatabase.db**: kezdeti adatbázis, ha a telefon magyar nyelvre van állítva
  - > **res/raw-hu\_rHU-long-trackball/initialDatabase.db**: kezdeti adatbázis, ha a telefon magyar nyelv van állítva, a kijelző szélesvásznú és van trackball



# Néhány hasznos metódus

- **getFilesDir()**
  - > Visszaadja az alkalmazás védett tárterületére mutató fájl objektumot ( data/data/[Package név] )
- **getDir()**
  - > Létrehoz vagy megnyit egy könyvárat az internal storage-en belül
- **deleteFile()**
  - > Fájlt töröl az internal storage könyvárban
- **fileList()**
  - > Egy String tömbben visszaadja az internal storage-ben lévő fájlok neveit

# Futási idejű engedélyek

# Mikor van rá szükség?

- Felhasználót „veszélyeztető” műveletek
- Engedély kérés régebben:

- > Manifest engedélyek:

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

- Új Permission modell Android 6 óta:
  - > Veszélyes engedélyeket futási időben kell kérni

# Engedély ellenőrzése

- Check permission:

```
ContextCompat.checkSelfPermission(thisActivity,  
    Manifest.permission.WRITE_CALENDAR)
```

- Engedély kérés Activity-ből:

- > Ellenőrizni, hogy megvan-e már az engedélye
- > Felhasználó tájékoztatása az engedély kérés okáról

# Engedély típusok

- Típusok
  - > Normal permissions
  - > Dangerous permissions
  - > <https://developer.android.com/guide/topics/security/permissions.html#normal-dangerous>
- A felhasználó visszavonhatja az engedélyeket a beállításokban bármikor
- További részletek:
  - > <https://developer.android.com/training/permissions/requesting.html>

# Permission kérés 1/2

```
private 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  
    }  
}
```

# 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()
            }
        }
    }
}
```

# Fájlkezelés - External storage



# Nyilvános lemezterület

- Lehet akár SD kártyán, akár belső (nem kivehető) memóriában
- Bárki által írható, olvasható a teljes fájlrendszer
- Amikor a felhasználó összeköti a telefont a számítógépével, és „*USB storage*” módra vált (mount), a fájlok hirtelen csak olvashatóvá válnak az alkalmazások számára
- Semmilyen korlátozás/tiltás nincs arra, hogy a nyilvános területen lévő fájljainkat a felhasználó letörölje, lemásolja vagy módosítsa!
  - > Amit ide írunk, az bármikor elveszhet

# Nyilvános lemezterület

- Legfontosabb tudnivalók
  - > Használat előtt ellenőrizni kell a tárhely elérhetőségét
  - > Fel kell készülni arra, hogy bármikor elérhetetlenné válik

```
val state: String = Environment.getExternalStorageState()

// sokféle állapotban lehet, nekünk kettő fontos:
when (state) {
    Environment.MEDIA_MOUNTED -> {
        // Olvashatjuk és írhatjuk a külső tárat
    }
    Environment.MEDIA_MOUNTED_READ_ONLY -> {
        // Csak olvasni tudjuk
    }
    else -> {
        // Valami más állapotban van, se olvasni,
        // se írni nem tudjuk
    }
}
```

# Nyilvános lemezterület

- Fájlok elérése a nyilvános tárhelyen 2.2 verziótól felfelé:

```
val filesDir: File = getExternalFilesDir(type: Int)
```

- type: megadhatjuk milyen típusú fájlok könyvtárát akarjuk használni, például:
  - > null: nyilvános tárhely gyökere
  - > DIRECTORY\_MUSIC: zenék, ahol az zenelejátszó keres
  - > DIRECTORY\_PICTURES: képek, ahol a galéria keres
  - > DIRECTORY\_RINGTONES: csengőhangok, ez is hang fájl, de nem zenelejátszóban akarjuk hallgatni
  - > DIRECTORY\_DOWNLOADS: letöltések default könyvtára
  - > DIRECTORY\_DCIM: a kamera ide rakja a fényképeket
  - > DIRECTORY\_MOVIES: filmek default könyvtára

# Nyilvános lemezterület

- Média típusonként külön alapértelmezett könyvtárak
- Így az azokat lejátszó/kezelő alkalmazásoknak nem kell az egész lemezt végigkeresni, csak a megfelelő könyvtárakat
- Indexelésüket a MediaScanner osztály végzi
  - > Ez mindenhol keres, és ha a talált média fájlok nem default könyvtárban vannak, akkor megpróbálja kategorizálni őket kiterjesztésük és MIME típusuk szerint
  - > Ha nem szeretnénk beengedni egy könyvtárba, akkor egy üres fájlt kell elhelyezni, melynek neve: ".nomedia"
    - Így például egy alkalmazás által készített fotók nem fognak látszódni a galériában
  - > A megfelelő default könyvtárba rakjuk az alkalmazásunk által létrehozott fájlokat, ha meg akarjuk osztani a userrel
- ***android.permission.WRITE\_EXTERNAL\_STORAGE***

# Nyilvános lemezterület

## Android 2.2 alatt

- External storage elérése: `getExternalStorageDirectory()`
- Ez a gyökérre ad referenciát
- Innen az `Android/data/[Package név]/files` könyvtárat használjuk
- Nincsenek konstansok a média típusokhoz, tudnunk kell hogy melyik default könyvtárnak mi a neve, és „kézzel” kell beleraknunk a média fájlokat
  - > `PL DIRECTORY_MUSIC = „Music/”`
- A 2.1-es verzió még nem halt ki teljesen, érdemes felkészítenünk az alkalmazásunkat rá! (2.2 alatt jelenleg az eszközök 2 százaléka)

# File írás (Java módra)

```
private fun writeFile(data: String) {  
    val file = File( "text.txt")  
  
    var outputStream: FileOutputStream? = null  
    try {  
        outputStream = FileOutputStream(file)  
        outputStream.write(data.toByteArray())  
        outputStream.flush()  
    } catch (e: IOException) {  
        e.printStackTrace()  
    } finally {  
        if (outputStream != null) {  
            try {  
                outputStream.close()  
            } catch (e: IOException) {  
                e.printStackTrace()  
            }  
        }  
    }  
}
```

# File írás (röviden)

```
private fun writeFile(data: String) {  
    val file = File(Environment.getExternalStorageDirectory(),  
                    "text.txt")  
  
    try {  
        file.writeText(data)  
    } catch (e: IOException) {  
        e.printStackTrace()  
    }  
}
```

# File olvasás (Java módra)

```
private fun readFile(): String? {  
    val file = File(Environment.getExternalStorageDirectory(), "text.txt")  
  
    var reader: BufferedReader? = null  
    try {  
        reader = BufferedReader(InputStreamReader(FileInputStream(file)))  
        val builder = StringBuilder()  
        while (true) {  
            val line: String? = reader.readLine()  
            if (line != null) {  
                builder.append(line)  
                builder.append("\n")  
            } else {  
                return builder.toString()  
            }  
        }  
    } catch (e: IOException) {  
        e.printStackTrace()  
    } finally {  
        if (reader != null) {  
            try {  
                reader.close()  
            } catch (e: IOException) {  
                e.printStackTrace()  
            }  
        }  
    }  
    return null  
}
```



# File olvasás (röviden)

```
private fun readFile(): String? {  
    val file = File(Environment.getExternalStorageDirectory(),  
                    "text.txt")  
  
    try {  
        return file.readText()  
    } catch (e: IOException) {  
        e.printStackTrace()  
        return null  
    }  
}
```

# Backend as a Service

Adatkezelés a felhőben



# Mi található a szerver oldalon?

## 1. Saját implementáció

- > PHP, Java, .NET, Node.JS, etc.
- > Software as a service (SaaS)

## 2. Felhő szolgáltatás használata

- > Platform as a Service: saját implementáció futtatása egy cloud megoldásban
  - OpenShift, Heroku, Azure, Amazon, etc.
- > Backend as a Service: háttér szolgáltatások használata, melyek elrejtik a bonyolult DB műveleteket és kommunikációt  
*Parse, Kumulus, Backendless*

# Saját szerver oldali implementáció

- Java
  - > Spring
  - > JAX-RS (Jersey)
- Server:
  - > Tomcat, GlassFish, JBoss, etc.
- DataBase:
  - > MySQL, PostgreSQL, Oracle, etc.
- Example:
  - > <http://babcomaut.aut.bme.hu:10080/RESTServerDemo/rest/api/time>
- More information:
  - > <http://www.ibm.com/developerworks/library/x-springandroid/>

# JAX-RS Jersey example – server code!!!

```
@Path("/api")
public class RestTest {
    // This method is called if TEXT_PLAIN is request
    @GET
    @Path("/time")
    @Produces(MediaType.TEXT_PLAIN)
    public String sayPlainTextHello() {
        return new Date(System.currentTimeMillis()).toString();
    }

    // This method is called if XML is request
    @GET
    @Produces(MediaType.TEXT_XML)
    public String sayXMLHello() {
        return "<?xml version=\"1.0\"?>" + "<hello> Hello Jersey" + "</hello>";
    }

    // This method is called if HTML is request
    @GET
    @Produces(MediaType.TEXT_HTML)
    public String sayHtmlHello() {
        return "<html> " + "<title>" + "Hello Jersey" + "</title>"
            + "<body><h1>" + "Hello Jersey" + "</body></h1>" + "</html> ";
    }
}
```

# BaaS szolgáltatások használata

- BaaS: Backend as a Service
  - > Felhasználó kezelés
  - > Perzisztencia, adatmentés, táblák
  - > File kezelés
  - > Verziókezelés
  - > Analytics
  - > Kód generálás
  - > Media streaming
  - > Geolocation
  - > Közösségi hálózati integráció
  - > További szolgáltatások, pl.: Push notification
- Firebase: <https://firebase.google.com/>
- Kumulos: <http://www.kumulos.com/>
- Backendless: <http://backendless.com/>

# BaaS Demo - Firebase



- Firebase
  - > Persistence
  - > Push notifications
  - > Authentication
  - > Analytics, crash reporting
- További részletek:
  - > <https://firebase.google.com/>



# Firestore fő funkciók

- [https://www.youtube.com/watch?list=PLl-K7zZEsYlM0F\\_07layrTntevxtbUxDL&time\\_continue=68&v=U5aeM5dvUpA](https://www.youtube.com/watch?list=PLl-K7zZEsYlM0F_07layrTntevxtbUxDL&time_continue=68&v=U5aeM5dvUpA)
- Real time adatbázis: JSON alapú NoSQL tárolás
  - > Perzisztens
  - > Eseményvezérelt, minden változásról értesítés
- Cloud Firestore (új generációs real-time adatbázis fejlett lekérdezés támogatással)
- Authentikáció:
  - > E-mail/közösségi hálózatok/egyedi
- Storage:
  - > file/kép tárolás
- Crash reporting
- Analytics
- Notifications
- ...



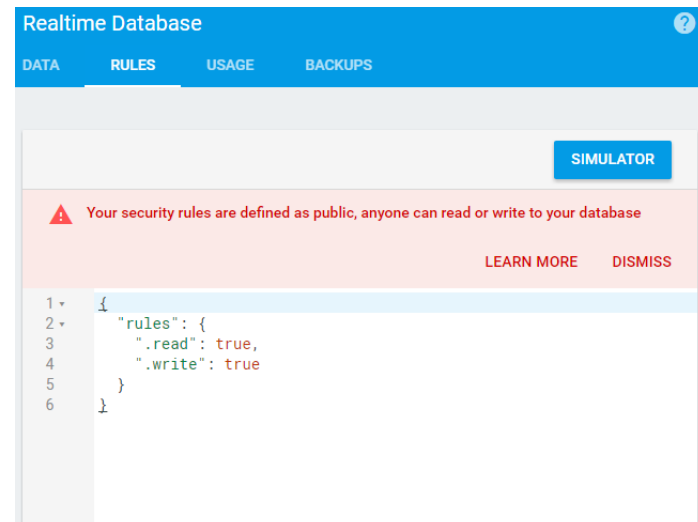
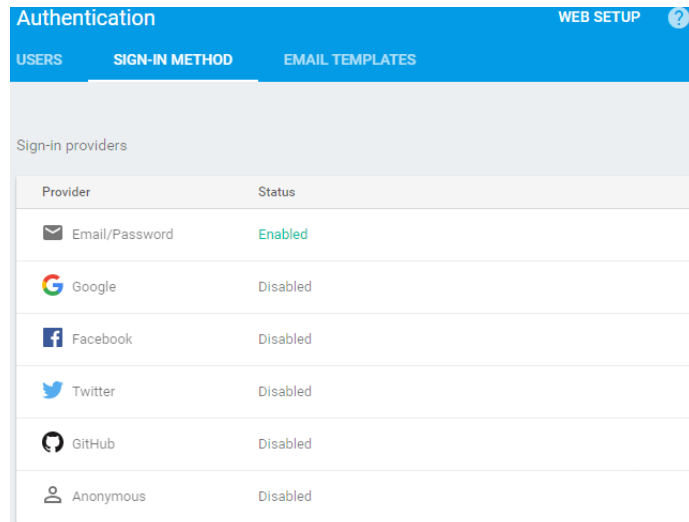
# Firebase – Első lépések

- Új alkalmazás a Firebase console-ban: <https://console.firebase.google.com>
- Gradle Dependency:  

```
implementation 'com.google.firebase:firebase-core:11.6.0'
```

  
Aktuális verzió: <https://firebase.google.com/docs/android/setup>
- Manifest permission:  

```
<uses-permission android:name="android.permission.INTERNET" />
```
- E-mail bejelentkezés engedélyezése
- Adat hozzáférési szabályok engedélyezése



# Insert művelet

```
val dbRef = FirebaseDatabase.getInstance().reference.child("posts")
val key = dbRef.push().key
Post newPost = Post(123, "important", "body")
FirebaseDatabase.getInstance().dbRef.
    child(key).setValue(newPost)
```

# Lekérdezés – feliratkozás új adataira

```
val ref = FirebaseDatabase.getInstance().getReference("posts")
ref.addChildEventListener(object : ChildEventListener {
    override fun onChildAdded(dataSnapshot: DataSnapshot?, s: String?) {
        val post = dataSnapshot?.getValue(Post::class.java)
        post?.let {
            postsAdapter.addPost(it, dataSnapshot.key)
        }
    }

    override fun onChildChanged(dataSnapshot: DataSnapshot?, s: String?) {
    }

    override fun onChildRemoved(dataSnapshot: DataSnapshot?) {
        postsAdapter.removePostByKey(dataSnapshot.key)
    }

    override fun onChildMoved(dataSnapshot: DataSnapshot?, s: String?) {
    }

    override fun onCancelled(databaseError: DatabaseError?) {
    }
}))
```

# Regisztráció

```
FirebaseAuth.getInstance().createUserWithEmailAndPassword(
    etEmail.text.toString(), etPassword.text.toString()
).addOnCompleteListener {
    if (it.isSuccessful) {
        val user = it.result.user

        user.updateProfile(
            UserProfileChangeRequest.Builder().setDisplayName(
                userNameFromEmail(user.email!!)).build()
        )

        Toast.makeText(this@LoginActivity, "Register success",
            Toast.LENGTH_SHORT).show()
    } else {
        Toast.makeText(this@LoginActivity, "Error: "+
            it.exception?.message,
            Toast.LENGTH_SHORT).show();
    }
}.addOnFailureListener {
    Toast.makeText(this@LoginActivity,
        "Error: ${it.message}",
        Toast.LENGTH_SHORT).show();
}
```

# Bejelentkezés

```
FirebaseAuth.getInstance().signInWithEmailAndPassword(
    etEmail.text.toString(), etPassword.text.toString()
).addOnCompleteListener {
    if (it.isSuccessful) {
        startActivity(Intent(this@LoginActivity, MainActivity::class.java))
    } else {
        Toast.makeText(this@LoginActivity, "Error: "+
            it.exception?.message,
            Toast.LENGTH_SHORT).show();
    }
}.addOnFailureListener {
    Toast.makeText(this@LoginActivity,
        "Error: ${it.message}",
        Toast.LENGTH_SHORT).show();
}
```

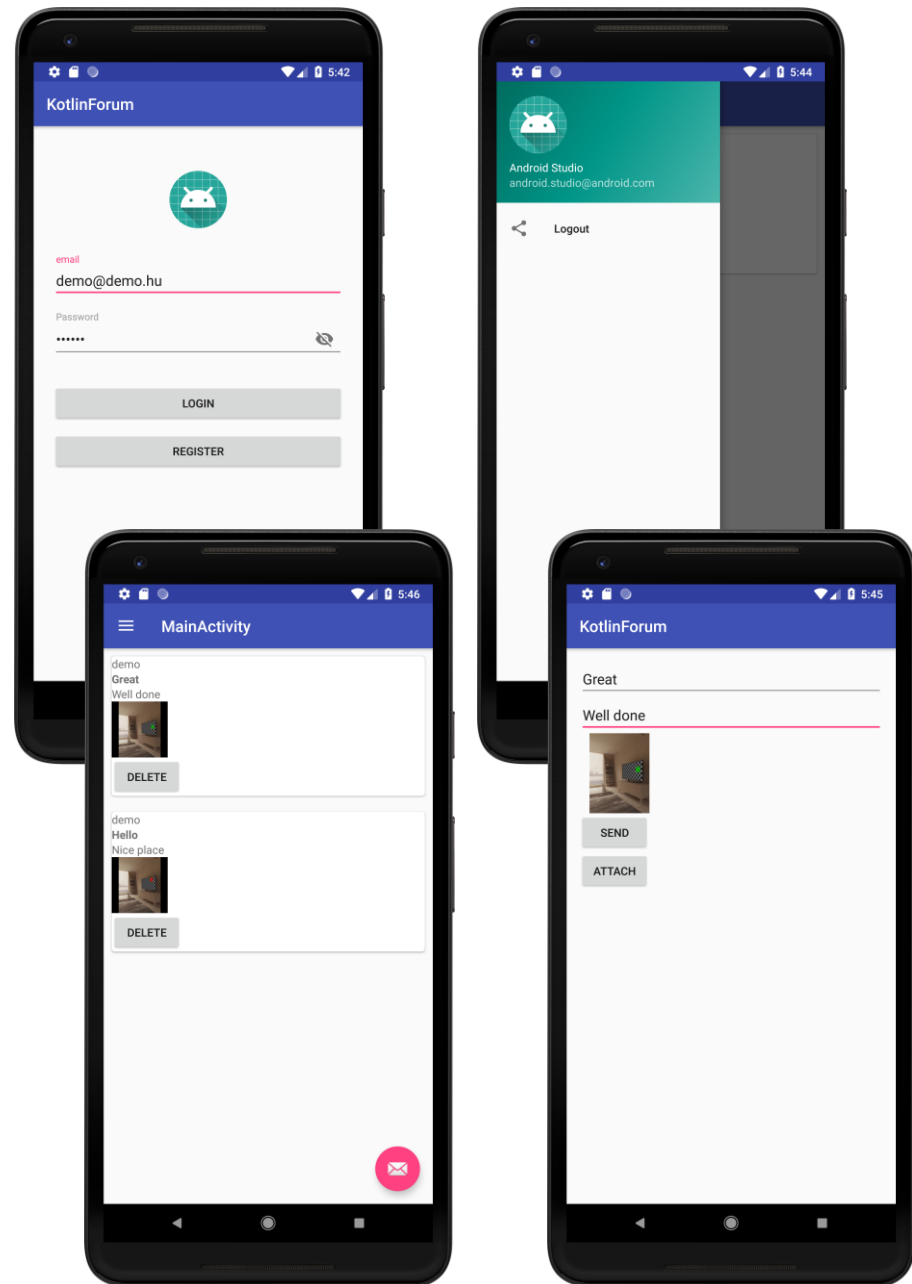
# Aktuális felhasználó elérése

```
val currUser: FirebaseUser =  
    FirebaseAuth.getInstance().currentUser
```

Csak egyszerűen 😊

# Gyakoroljunk

- Készítsünk egy forum alkalmazást
- Felhasználók kezelése
- Üzenetek real time megjelenítése
- Tartalom megjelenítése CardView-n
- NavigationDrawer alapú menü
- Kép feltöltés kameráról



# Android GCM / Firebase FCM

- Sokszor szükség lehet rá, hogy a szerver tájékoztassa a mobil klienseket valamilyen eseményről
- Jelenlegi eszközünk:
  - > Poll-ozás
  - > Kliens időközönként lekérdezi a szervertől, hogy van-e számára üzenet
  - > Lassú és nem real-time
- Fordított irány: valahogy a szervernek kéne tájékoztatni a klienseket
- Megoldás:
  - > (régi) GCM (korábban Android Cloud to Device Messaging Framework)
  - > FCM: Firebase Cloud Messaging
- Szerver és kliens oldali implementáció szükséges
- Regisztráció és további információk a használati feltételekről:
  - > <http://developer.android.com/guide/google/gcm/index.html>

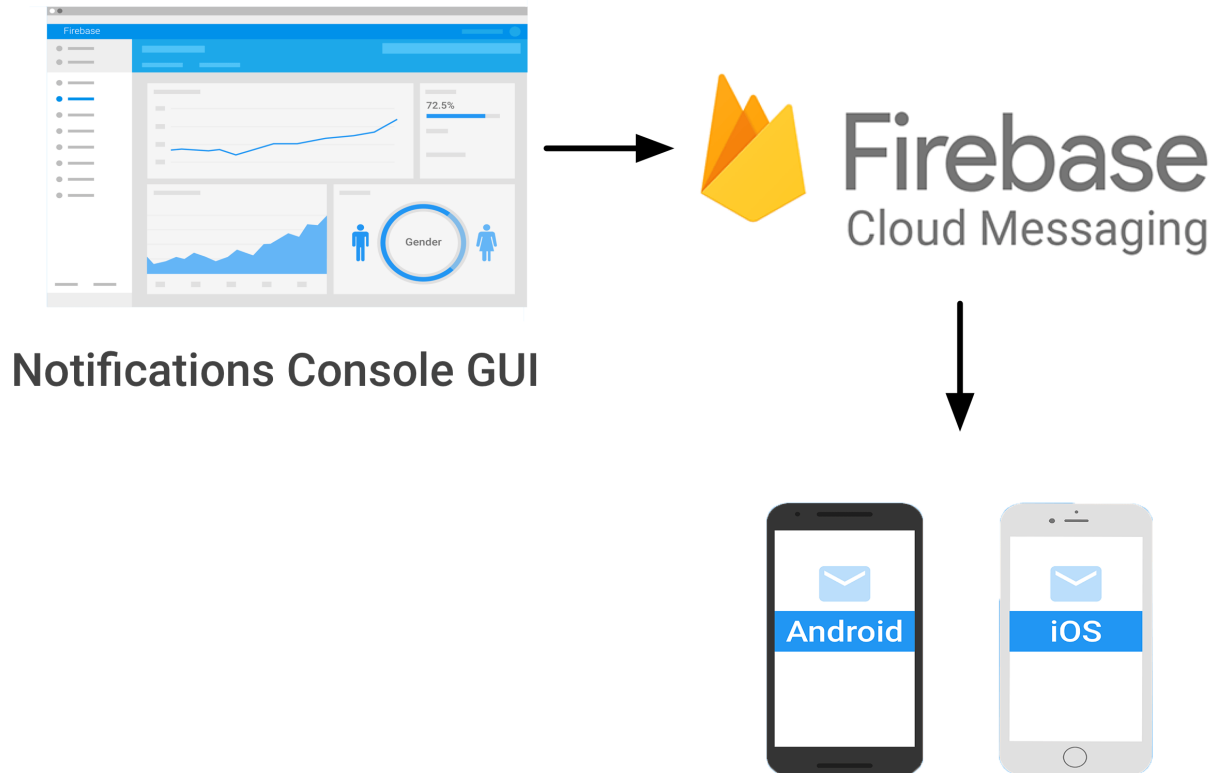




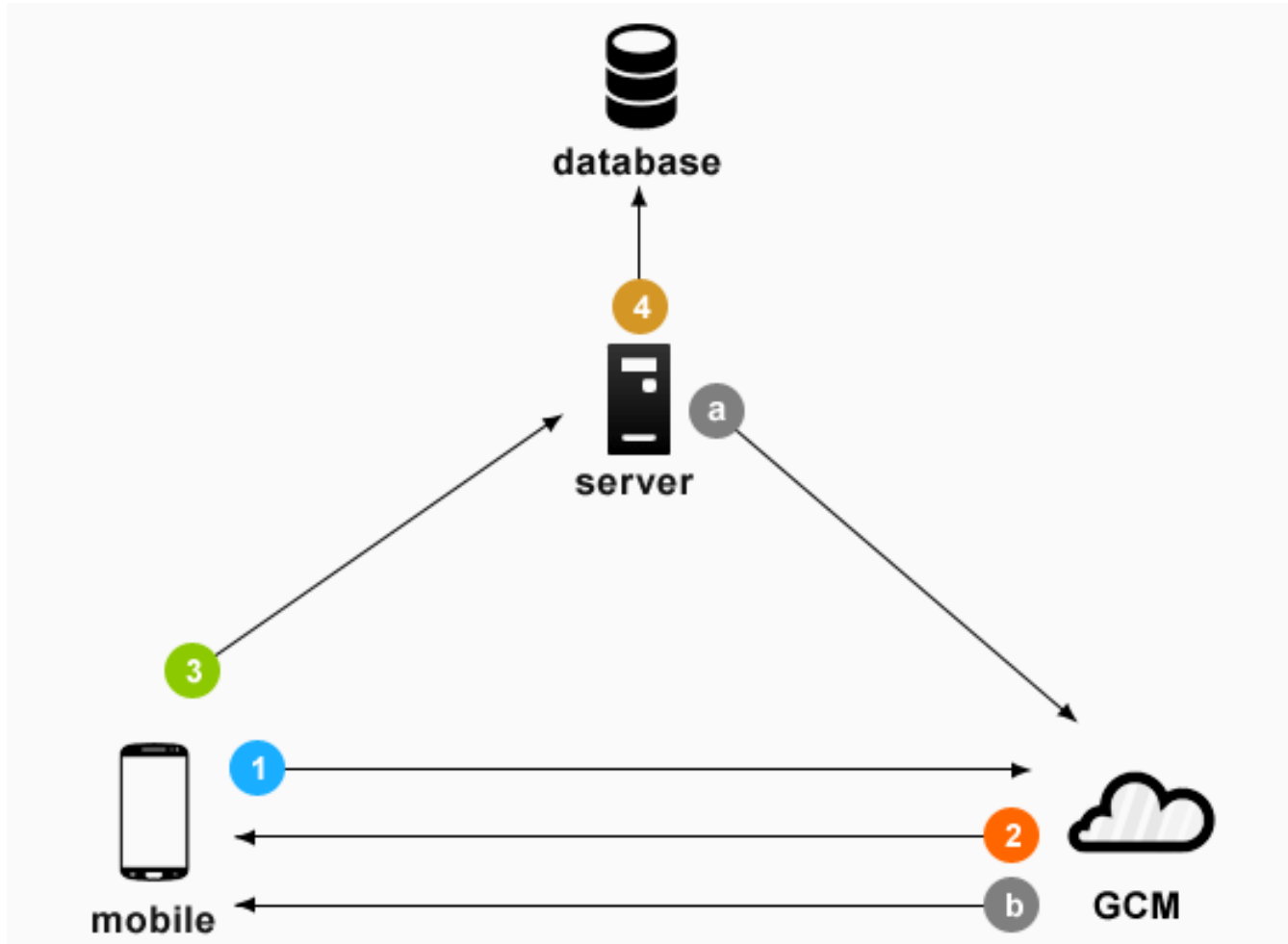
# Firebase FCM

- Cél: szerver -> kliens kommunikáció
- Rövid üzenetek a kliensek értesítésére
- Ingyenes használat
- FCM komponensek:
  - > Android készülék
  - > Application server
  - > FCM cloud

# Firestore Cloud Messaging (push notification)

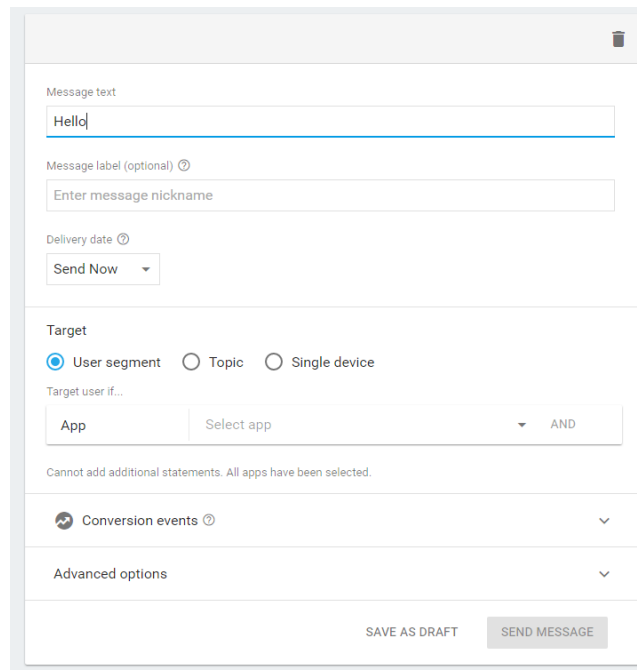


# GCM/FCM Architektúra

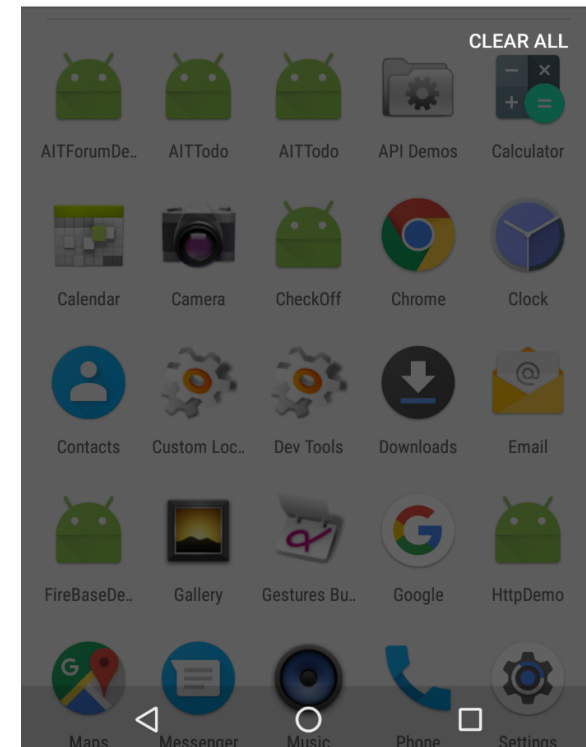
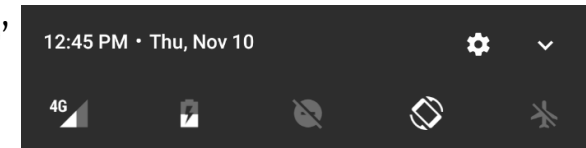


# Push értesítések kezelése

- Gradle dependency:
  - > implementation 'com.google.firebase:firebase-messaging:11.6.0'
- Console-ról tesztelhető
- Saját push receiver:
  - > <https://firebase.google.com/docs/cloud-messaging/android/receive>

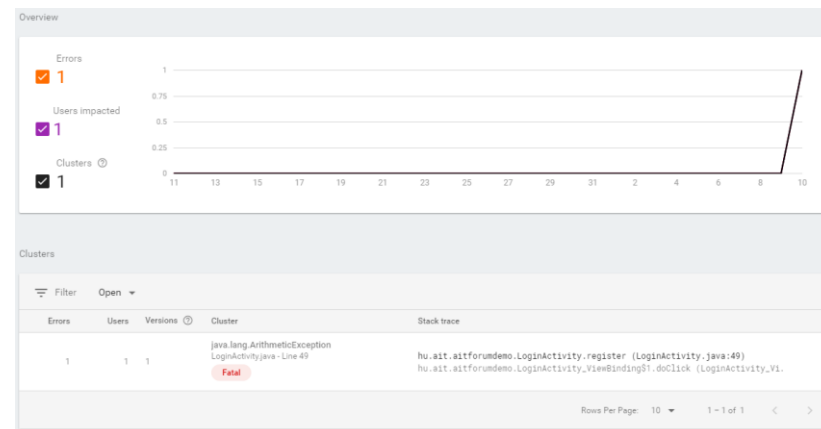


The screenshot shows the 'Compose message' interface in the Firebase console. It includes a 'Message text' field with 'Hello', a 'Message label (optional)' field with placeholder text 'Enter message nickname', and a 'Delivery date' dropdown set to 'Send Now'. Under the 'Target' section, 'User segment' is selected. Below this, there's a 'Select app' dropdown and an 'AND' connector. A note states 'Cannot add additional statements. All apps have been selected.' There are also expandable sections for 'Conversion events' and 'Advanced options'. At the bottom, there are 'SAVE AS DRAFT' and 'SEND MESSAGE' buttons.



# Crash reporting

- Gradle dependency:
  - > implementation 'com.google.firebase:firebase-crash:11.6.0'
- Kivételek automatikus kezelése Firebase-ben
- <https://firebase.google.com/docs/crashlytics/get-started?authuser=0>
- Egyedi log üzenetek:
  - > `FirebaseCrash.log("Button pressed")`



# Összefoglalás

- Egyszerű kulcs-érték táár: SharedPreferences
- File-kezelés, belső és külső tárterület
- Firebase Backend as a Service
  - > Felhasználók kezelése
  - > Real-time adatkezelés
  - > Kamera kezelés
  - > Képek feltöltése
  - > Crash reporting
  - > Push notification

# A következő alkalommal...

- Rövid és hosszabb távú kapcsolatok bemutatása (NFC, Bluetooth, UDP, TCP/IP socket)
- HTTP kapcsolatok kezelése
- Tipikus adatformátumok és azok feldolgozási lehetőségei
- AsyncTask és BroadcastReceiver komponensek
- Aszinkoron kommunikáció helyes kezelése
- REST API-k hatékony használata
- Aszinkron Kotlin nyelvi elemek
- Hatékony hálózati kommunikációs könyvtárak

# Köszönöm a figyelmet!



*[peter.ekler@aut.bme.hu](mailto:peter.ekler@aut.bme.hu)*



**AutSoft**