Android

File kezelés, Firebase, Helymeghatározás

Dr. Ekler Péter

peter.ekler@aut.bme.hu

BME AUT

Department of
Automation and
Applied Informatics

Tartalom

- Activity állapot elmentése
- File kezelés
- Firebase Backend as a Service
  - Felhasználók kezelése
  - Real-time adatkezelés
  - Képek feltöltése
  - Crash reporting
  - Push notification
- Helymeghatározás

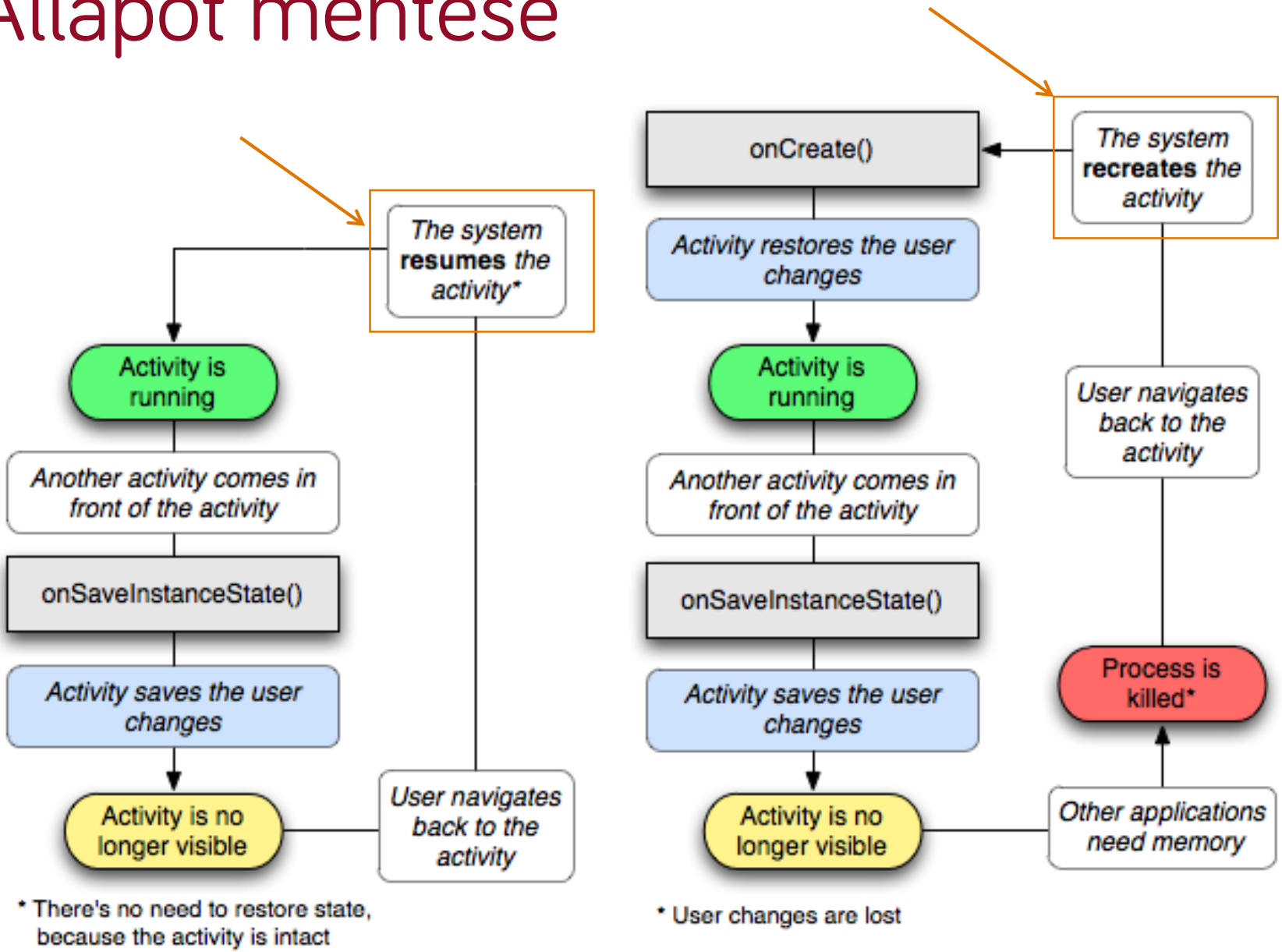
Állapotmentés

Activity visszaállításához szükséges adatok mentése

Állapot mentése

- Ha egy Activity *Paused* (részben takarja valami) vagy *Stopped* (egyáltalán nem látszik) állapotban van, akkor a memóriában tárolódik minden hozzá tartozó adat
- Folytatódáskor minden visszaáll „magától”, mert még ugyanaz a processz fut
- Azonban ha kevés memória miatt az Android felszabadította a területét, akkor a **processz meghal**, és vele együtt **minden elvész a memóriaterületről** (osztály tagváltozói, UI állapot információk, stb...)
- Újra előtérbe kerüléskor az *onCreate()* hívódik, gyakorlatilag újraindul az Activity
- **Képernyő elforgatáskor** is megsemmisül és újraépül az Activity!
- **A fejlesztőnek kell gondoskodnia az állapot mentéséről! – onSaveInstanceState()**

Állapot mentése



Állapot mentése

- Ezért kap az *onCreate()* paraméterként egy *savedInstaceState* nevű Bundle-t

```
@Override
public void onCreate(Bundle savedInstanceState) {...}
```

- Itt kell újra beállítani a változókat és a UI elemeket a kapott ***Bundle*** alapján
- Amit az ***onSaveInstanceState()*** -ben töltöttünk fel
- Ez az egész mechanizmus láthatatlan a felhasználó számára
  - Ha nem csináljuk meg, akkor csak azt veszi észre, hogy néha máshogy tér vissza az alkalmazás mint ahogy otthagyta
  - Emiatt **alapkövetelmény hogy a fejlesztő lekezelje az állapotmentést**, és mindig úgy folytatódjon az app, ahogy háttérbe került!

onSaveInstanceState()

- Az ***onSaveInstanceState()*** lefutása nem egyértelmű
  - Nem biztos hogy egyáltalán lefut, a Vissza gomb megnyomása esetén például az Android feltételezi, hogy a felhasználó ki akart lépni, így nincs szükség állapotmentésre (*onBackPressed()*-ben felülírható)
  - Nem meghatározott, hogy az *onPause()* előtt vagy után fut le (az biztos, hogy az *onStop()* előtt)
- Emiatt rossz gyakorlat perzisztens adat mentéséről (pl. adatbázisba, fájlba írás) gondoskodni az *onSaveInstanceState()*-ben! Helyette az *onPause()*-ban intézzük el, az biztosan lefut minden esetben
- A képernyő elforgatásakor az Activity elpusztul és újra létrejön (mert újra kell építenie a UI-t), ezzel jól tesztelhető az állapotmentés
  - A user szereti forgatni a telefont, emiatt is erősen ajánlott a helyes lekezelése!

onSaveInstanceState()

- Az onCreate()-ben hívott **`super.onCreate(savedInstanceState)`** magától visszaállít néhány dolgot, de nem mindent! Amit igen:
  - Majdnem minden UI Widget alapból visszáll
    - Pl. EditText, Checkbox, stb
    - **DE**: a Spinner például nem őrzi meg, hogy mi volt kiválasztva!
    - Nincs dokumentálva minden, ha kritikus akkor ki kell próbálni!
    - Csak akkor, ha van id-ja! (`android:id`)
  - Ezen kívül semmi mást nem állít vissza magától
  - UI elemre beállíthatjuk, hogy ne állítsa vissza az állapotát – **`android:saveEnabled="false"`**

Állapot mentése - HowTo

1. *onSaveInstanceState()*-ben a paraméterként kapott Bundle-be beleírunk mindent, ami a UI-on megjelenik, vagy egyéb okból szükséges visszaállítani
   - Listák aktuálisan kiválasztott elemei
   - Tagváltozók, amik futás közben változhattak
   - Stb…

2. *onCreate()*-ben visszatöltjük az értékeket, ha a kapott savedInstanceState nem null (akkor null, ha nem volt állapotmentés)
   - Ha nem az *onCreate()*-et akarjuk telenyomni a visszaállító kóddal, akkor tegyük az *onRestoreInstanceState()*-be, ami az *onCreate()* után hívódik

Állapotmentés Példa

```
class MainActivity : Activity() {
    private var string: String? = null
    private val KEY_STRING = "str"

    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        string = "A tagváltozó kezdeti értéke változhat a futás során!"
        if (savedInstanceState != null) {
            if (savedInstanceState.containsKey(KEY_STRING)) {
                string = savedInstanceState.getString(KEY_STRING)
            }
        }
    }

    override fun onSaveInstanceState(outState: Bundle) {
        outState.putString(KEY_STRING, string)
        super.onSaveInstanceState(outState)
    }
}
```

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
  - Bármi ami 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 intenal 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()
            }
        }
    }
}
```

Futási idejű engedélyek

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

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/

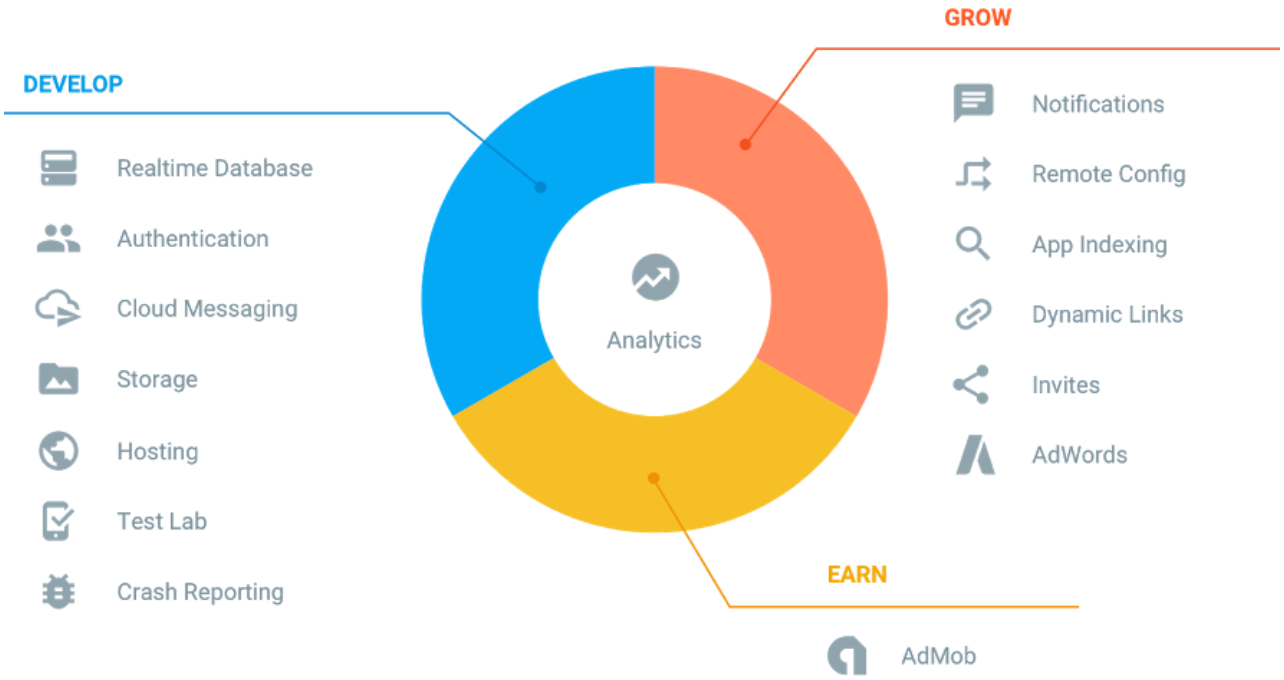
JAX-RS Jersey example – server code!!!

```
@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/

Firebase fő funkciók

- https://www.youtube.com/watch?list=PLl-K7zZEsYLmOF_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:

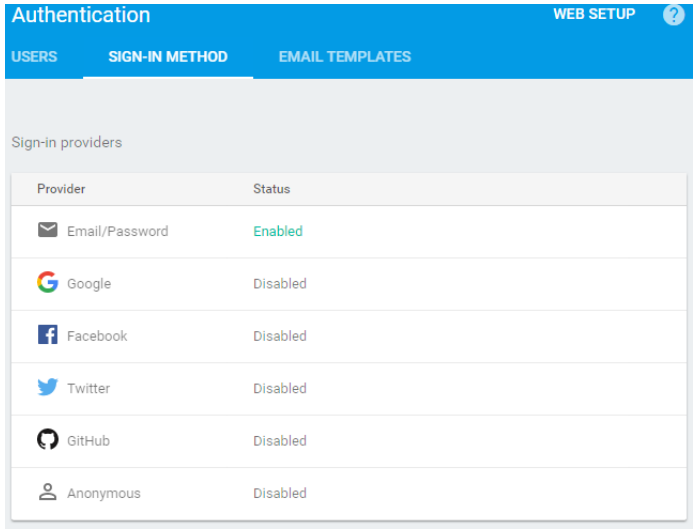
```
implementation 'com.google.firebase:firebase-core:11.6.0'
```

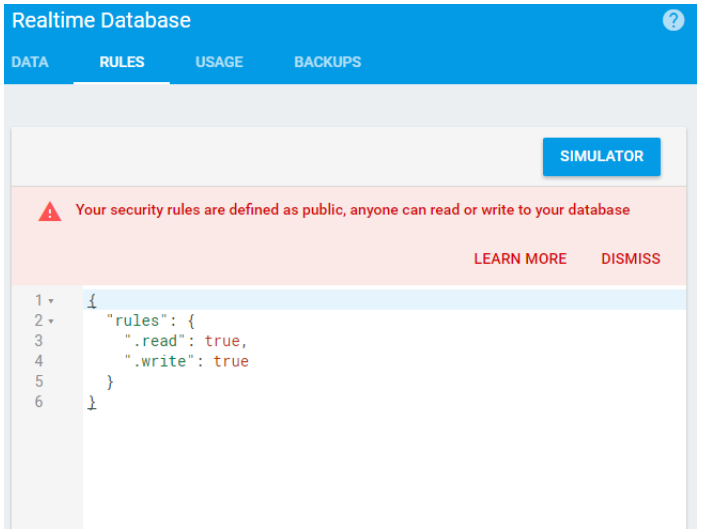
  Aktuális verzió: https://firebase.google.com/docs/android/setup

- 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 adatra

```
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

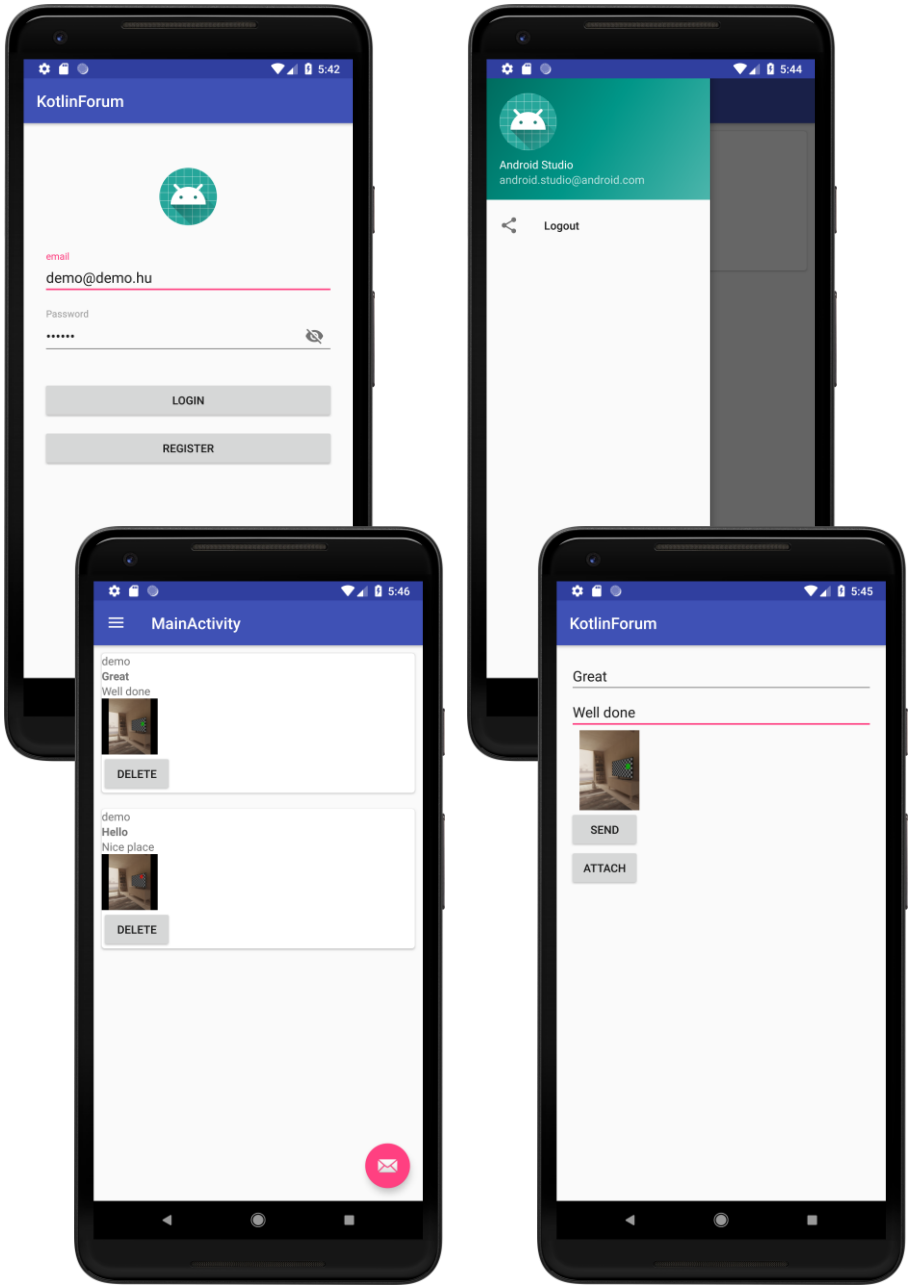
```
val currUser: FirebaseUser =
  FirebaseAuth.getInstance().currentUser
```

Csak egyszerűen ☺

Gyakoroljunk

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

https://github.com/peekler/AIT2018Summer/tree/master/AITForumApp



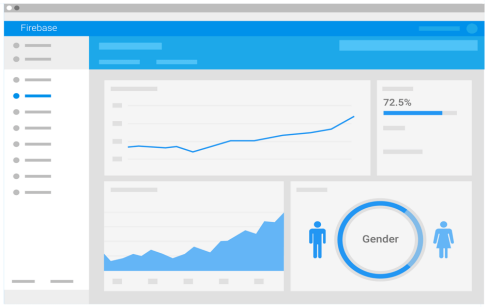
Android GCM / Firebase FCM

- 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

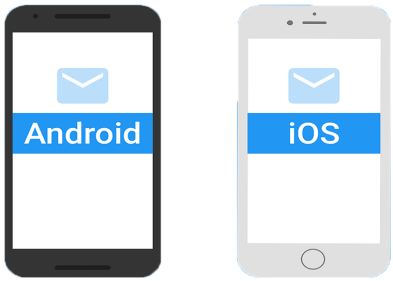
Firebase FCM

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

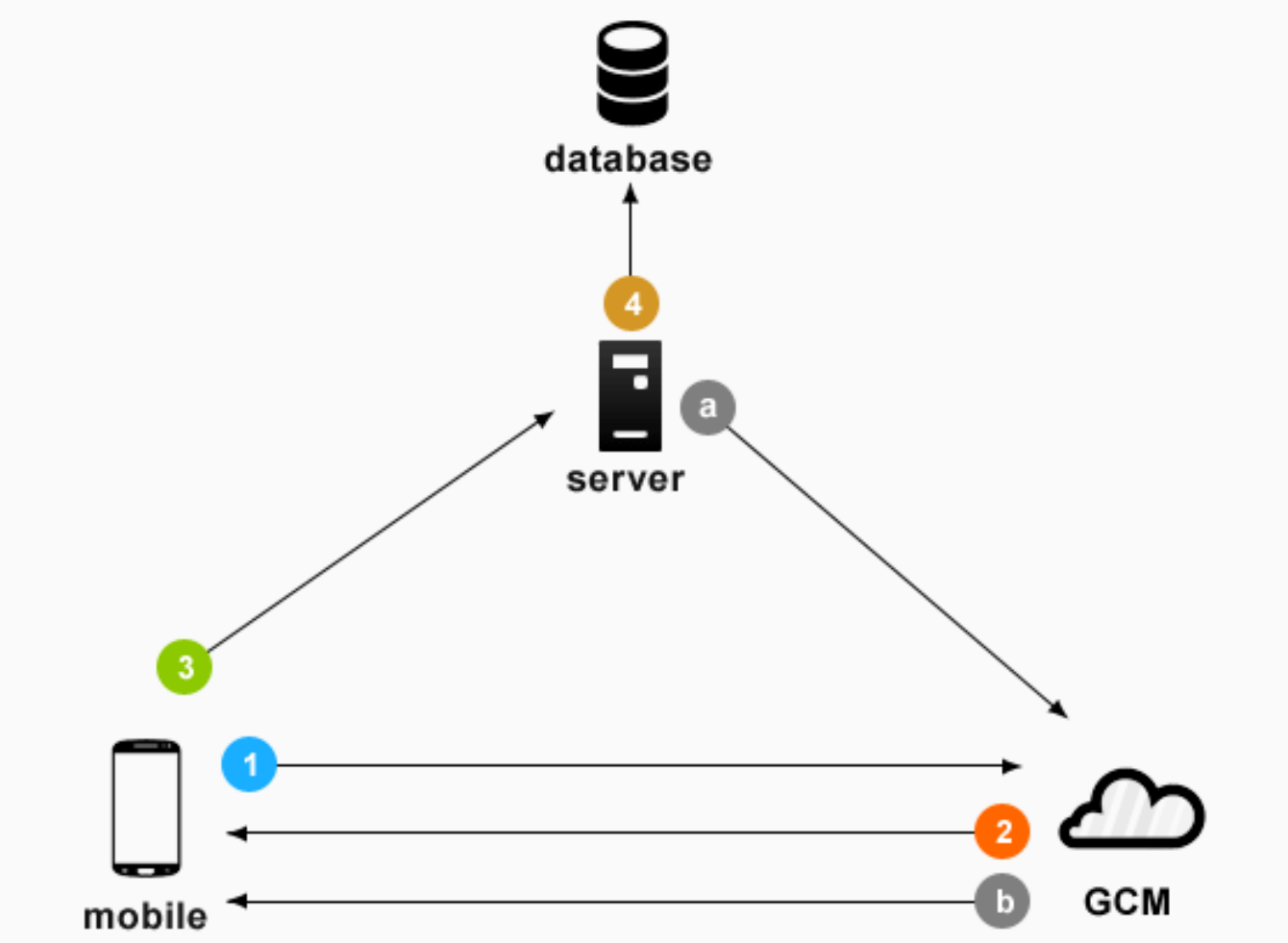
Firebase Cloud Messaging (push notification)



Notifications Console GUI

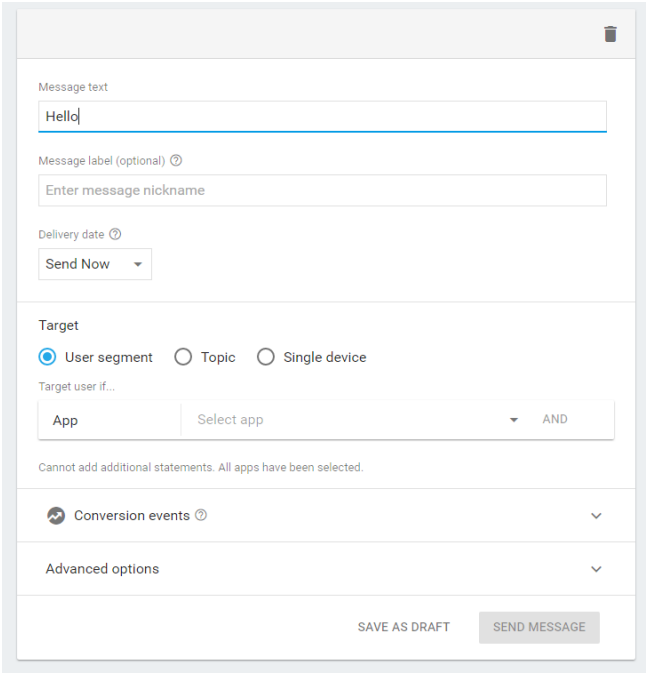


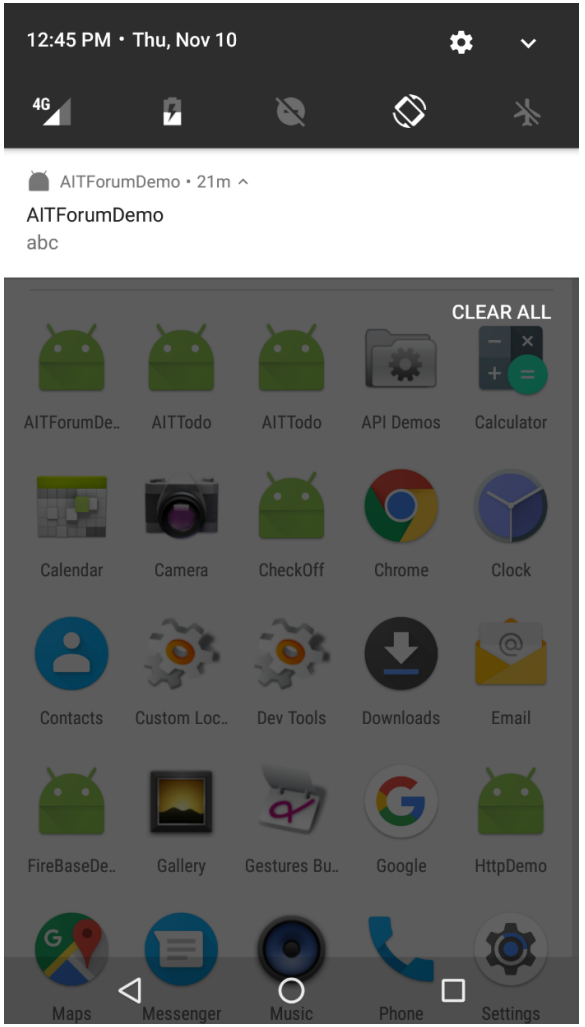
GCM/FCM Architektúra



Push értesítések kezelése

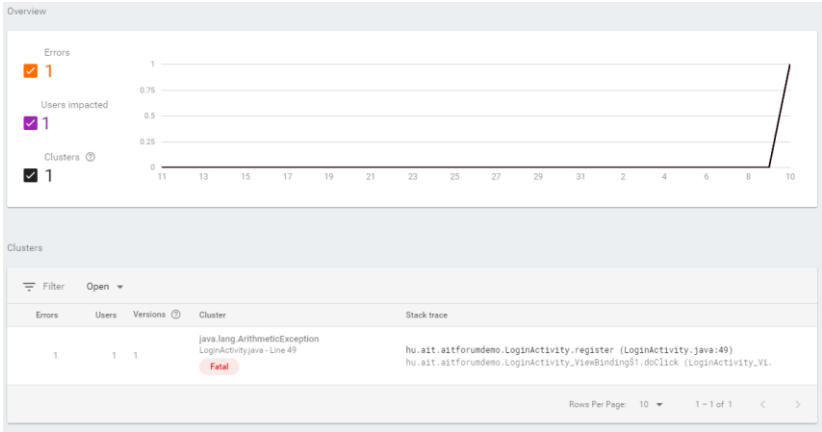
- Gradle dependency:
  - implementation 'com.google.firebase:firebase-messaging:11.6.0'
- Console-ről tesztelhető
- Saját push receiver:
  - https://firebase.google.com/docs/cloud-messaging/android/receive





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")



Helymeghatározás

Helymeghatározás

- Android által támogatott módok:
  - GPS
  - Hálózat: Android's Network Location Provider
- GPS pontos, de csak kültéren működik és magas az energiaigénye
- Hálózat alapú:
  - Cella alapú helymeghatározás
  - WiFi alapú helymeghatározás
  - Gyors, de gyakran pontatlan
- Egy időben akár mindkét módszert is alkalmazhatjuk

Nem triviális a pozíció meghatározása

- GPS, mobil hálózat és WiFi alapú metódus is adhat egyidőben információt, de figyelembe kell venni a következőket:
  - Pontosság
  - Sebesség
- A felhasználó folyamatos mozgásban lehet, sokszor szükség van becslésre
- Előfordulhat, hogy egy frissen kapott pozíció pontatlanabb, mint a 10 másodperccel ezelőtti!

Location engedély beállítása

- Manifest állományba új permission

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

- `ACCESS_COARSE_LOCATION`: csak hálózat alapú helymeghatározás engedély
- `ACCESS_FINE_LOCATION`: GPS és hálózat alapú helymeghatározás engedély

Android Location API 1/2

- Callback függvényekben érkezik az információ
- `LocationManager` rendszerszolgáltatás lekérése:

```
var locationManager: LocationManager =
getSystemService(Context.LOCATION_SERVICE) as LocationManager
```

- Folyamatos frissítés kérése egy `LocationListener` implementáció átadásával

```
// provider, mintime, mindistance, listener
locationManager.requestLocationUpdates(Locati
onManager.NETWORK_PROVIDER, 0, 0,
locationListener)
```

Android Location API 2/2

- requestLocationUpdates() paraméterei:
  - Provider típus (GPS vagy hálózat alapú)
    - Kétszer is meghívható más-más providerrel (GPS és hálzózat)
  - Minimum idő két frissítés között (0: lehető leggyakrabban)
  - Minimum távolság két frissítés között (0: lehető leggyakrabban)
  - LocationListener implementáció
- LocationListener callback függvényei:
  - Új pozíció:

```
onLocationChanged(Location location)
```

  - Provider állapotváltozás:

```
onStatusChanged(String provider, int status, Bundle extras)
```

  - Provider használhatóvá vált:

```
onProviderEnabled(String provider)
```

  - Provider nem használható

```
onProviderDisabled(String provider)
```

LocationListener példa

```
var locationListener: LocationListener = object  : LocationListener {
    override fun onLocationChanged(location: Location?) {
    }

    override fun onStatusChanged(provider: String?,
      status: Int, extras: Bundle?) {
    }

    override fun onProviderEnabled(provider: String?) {
    }

    override fun onProviderDisabled(provider: String?) {
    }

}
```

Leiratkozás

- `removeUpdates(...)` függvény hívás minden létrehozott `LocationListener`-re
- Nagyon fontos, hogy **ne felejtsük el**!
- `locationManager.removeUpdates(locationListener)`

FusedLocationProvider

- Alkalmazások képesek megosztani a hely információkat egymással
- Google Play Services függőség:

```
implementation 'com.google.android.gms:play-services-location:11.6.0'
```

- FusedLocationProviderClient

```
private val fusedLocationClient: FusedLocationProviderClient =
        LocationServices.getFusedLocationProviderClient(context)
```

Indítás és leállítás

```
fun startLocationMonitoring() {
    val locationRequest = LocationRequest()
    locationRequest.interval = 1000
    locationRequest.fastestInterval = 500
    locationRequest.priority = LocationRequest.PRIORITY_HIGH_ACCURACY


    fusedLocationClient.requestLocationUpdates(locationRequest,
            locationCallback, Looper.myLooper())

}

fun stopLocationMonitoring() {
    fusedLocationClient.removeLocationUpdates(locationCallback)
}
```

```
private var locationCallback: LocationCallback = object : LocationCallback() {
    override fun onLocationResult(locationResult: LocationResult) {
        super.onLocationResult(locationResult)
    }
}
```

Fontos tippek & javaslatok

- Ellenőrizzük, hogy a kapott pozíció jelentősen újabb-e
- Ellenőrizzük a kapott pozíció pontosságát (accuracy)
- Az előző pozícióból és a hozzá tartozó sebességből ellenőrizhető az új pozíció realitása
- Ellenőrizzük melyik provider-től származik az új pozíció
- 60 másodpercnél gyakoribb pozíciókérés sokszor felesleges
- Nem mindig kell GPS és hálózati providert egyszerre használni

Geocoding

- GPS koordináta postacímből
- Internet engedély szükséges

```
val geocoder = Geocoder(this, Locale.ENGLISH)
val streetAddress = "Blaha Lujza tér 1, Budapest"
var locations: List<Address>? = null
geocoder.getFromLocationName(streetAddress, 3)
```

Reverse Geocoding

- Cím GPS koordinátából
- Internet engedély szükséges

```
val location = …
val latitude = location.getLatitude()
val longitude = location.getLongitude()
val gc = Geocoder(this, Locale.getDefault())
var addrs: List<Address>? =
            gc.getFromLocation(latitude, longitude, 3)
```

Összefoglalás

- Activity állapot elmentése
- File kezelés
- Firebase Backend as a Service
  - Felhasználók kezelése
  - Real-time adatkezelés
  - Képek feltöltése
  - Crash reporting
  - Push notification
- Helymeghatározás

Kérdések