

Android

Architecture Components, JetPack, NFC, Library-k,
Útravaló

Dr. Ekler Péter
peter.ekler@aut.bme.hu



Department of
Automation and
Applied Informatics

Tartalom

- Architecture Components
- NFC
- Külső könyvtárak
- Statikus kódelemzés
- Android API érdekességek
- Útravaló

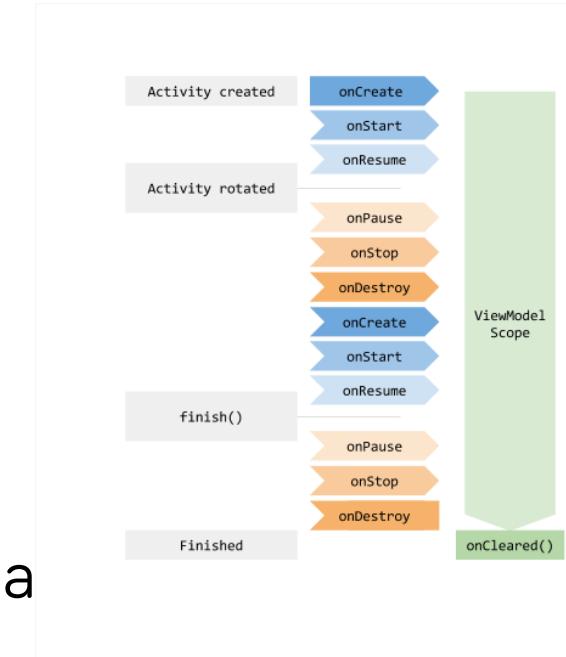
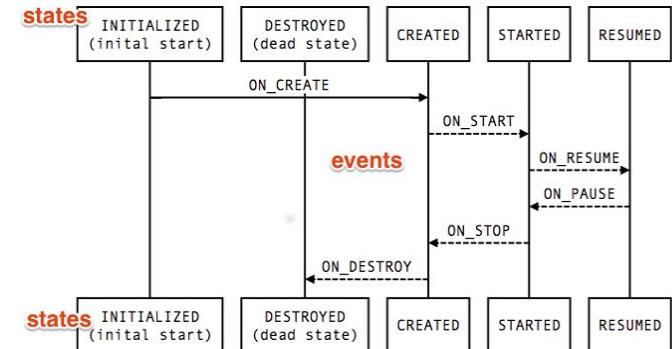
Alkalmazás architektúra

Miért kell beszélni az architektúráról?

- Mi kerüljön az Activity/Fragment osztályba?
- Hol tároljuk az adatokat?
- Hogyan kezeljük a konfiguráció változást (pl. képernyő elforgatás)?
- Hogyan érjük el, hogy az adatok változásakor frissüljön a UI?
- Milyen ORM-et használunk?
- Hogyan válasszuk szét az adatforrást az adatoktól?

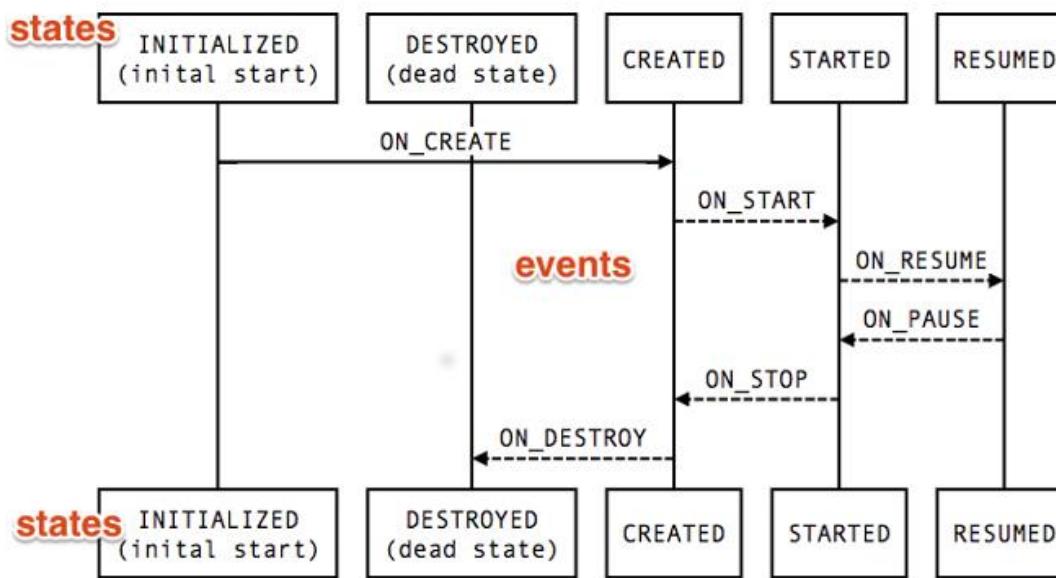
Architecture Components elemek

- Lifecycle / LifecycleObserver
 - > Lifecycle-függő komponensek, objektumok hozhatók létre
- LiveData
 - > Adat kezelő, megfigyelhetővé teszi az adatot
- ViewModel
 - > UI-on megjelenő adatok egyszerű kezelése, mely független a konfiguráció változástól
- Room Persistence Library
 - > Google saját ORM megoldása, a teljes SQLite képességeket kihasználja



Lifecycles

- Az Activity/Fragment életciklus közvetlen elérésére más komponensekből
 - > Feliratkozhatunk a változásokra, vagy elkérhetjük az aktuális állapotot (!)



Lifecycles

- Előnyök:

- > Rövidebb kód az életciklussal rendelkező osztályokban

```
override fun onStart() {  
    super.onStart()  
    compassListener.start()  
    gpsService.getUpdates()  
    mediaPlayer.resume()  
}
```

```
override fun onStop() {  
    super.onStop()  
    compassListener.stop()  
    gpsService.pauseUpdates()  
    mediaPlayer.stop()  
}
```

- > A komponens tudja, hogy mikor melyik függvényét kell futtatni
 - Nem felejtjük el meghívni őket amikor kell

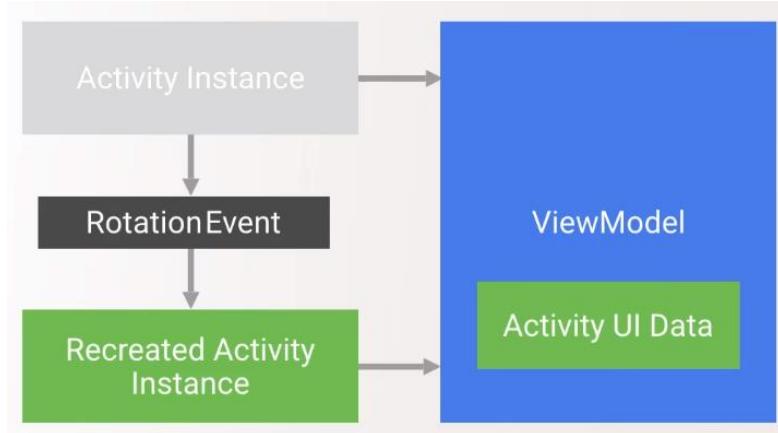
Lifecycles

```
class LifecycleAwareComponent : LifecycleObserver {
    @OnLifecycleEvent(Lifecycle.Event.ON_START)
    fun startDoingThings() {
        // ...
    }
    @OnLifecycleEvent(Lifecycle.Event.ON_STOP)
    fun stopDoingThings() {
        // ...
    }
}

...
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    lifecycle.addObserver(LifecycleAwareComponent())
}
```

ViewModel

- Felhasználó felület adatainak tárolására, életciklus tudatos módon
 - > Tetszőleges adatot tartalmazhat, ami a UI feltöltésére szolgál
 - > Túléli a konfiguráció változásokat, az Activity/Fragment újraindulását

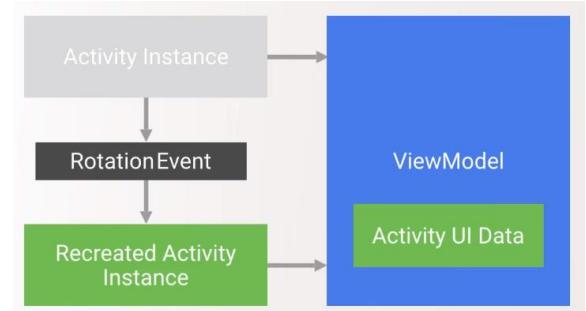


ViewModel

- Előnyök:
 - > Az Activity-nek a felelőssége így csak a megjelenítés
 - > A ViewModel tárolja (ideiglenesen) és menedzseli az adatokat

```
class UserViewModel : ViewModel() {  
    val user = User(name = "Sally", age = 25)  
}
```

```
class UserActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        val userViewModel =  
            ViewModelProviders.of(this).get(UserViewModel::class.java)  
    }  
}
```

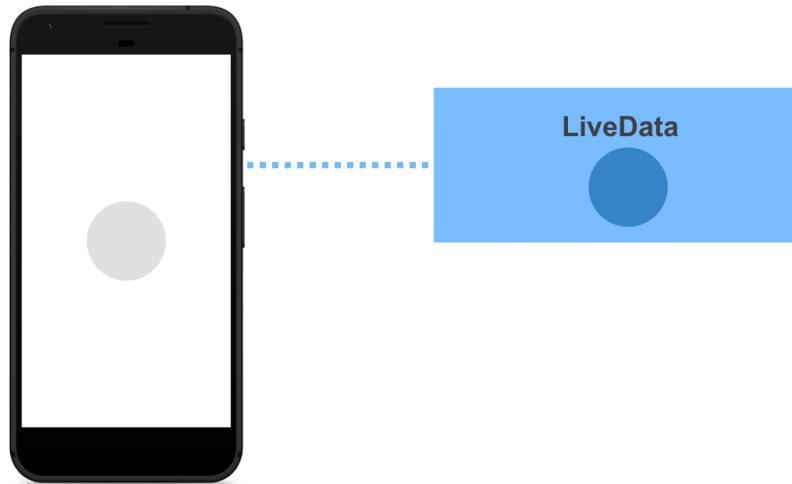


ViewModel

- Tippek: a ViewModel...
 - > ... soha ne tartson referenciát UI elemekre, vagy például egy Activity-re
 - Mivel hosszabb életű az Activity-nél, ez memória szivárgást jelentene
 - > ... nem helyettesíti a savedInstanceState-et
 - ViewModel: több adatot tárolhat
 - Tárolja a UI megjelenítéséhez szükséges összes adatot
 - savedInstanceState: túléli a konfiguráció változást, és még a teljes alkalmazás halálát is ha túl kevés a memória, de kis mennyiségű adatot tárol
 - Tároljon minimális mennyiségű adatot a UI visszaállításához (például ID-kat)

LiveData

- Lifecycle aware, observable data holder
 - > Életciklus tudatos, megfigyelhető adat tároló...
- Megfigyelhető:
 - > Becsomagol egy értéket, és a feliratkozott megfigyelőit értesíti, ha az érték megváltozott



LiveData

- Jellemzően egy ViewModel tartalmazza
 - > Így megfigyelhetjük az adatait, anélkül hogy a ViewModel hivatkozna az Activity-re

```
class UserViewModel : ViewModel() {  
    val user: LiveData<User> = ...  
}  
  
class UserActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        val userViewModel = ViewModelProviders.of(this)  
            .get(UserViewModel::class.java)  
  
        userViewModel.user.observe(this, Observer { user ->  
            // show user on UI  
        })  
    }  
}
```

LiveData

- Életciklus tudatos
 - > Csak akkor hívja meg a megfigyelőt, ha az aktív
 - Nem kap frissítést az Activity, ha épp nem látszik
 - Amikor újra láthatóvá válik, csak a legutolsó értéket kapja meg
 - > A feliratkozást automatikusan bontja, ha az átadott LifecycleOwner életciklusának vége lesz
 - Nem felejtjük el
 - Nincs memória szivárgás

```
userViewModel.user.observe(this, Observer { user ->
    // show user on UI
})
```

LiveData

- Példány létrehozása:

```
val user: MutableLiveData<User> = MutableLiveData<User>()
```

- Értékek frissítése UI szálról:

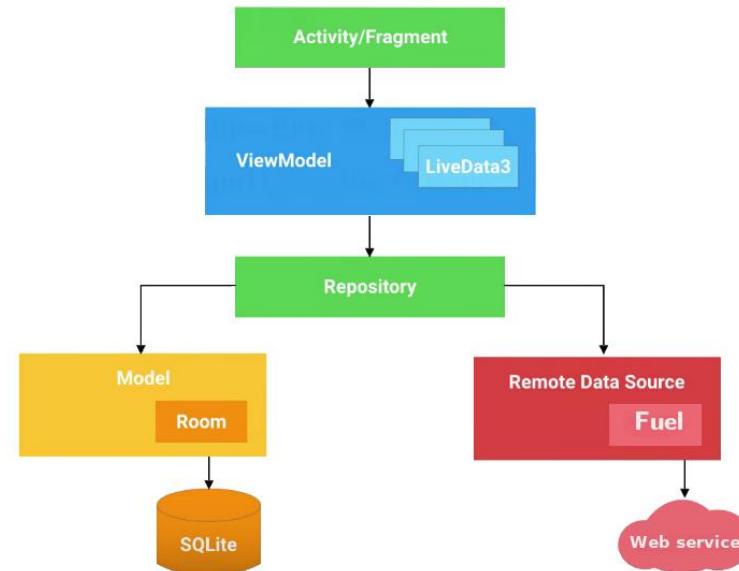
```
user.setValue(User("Ann", 56))  
user.value = User("Jim", 41)
```

- Értékek frissítése háttérszálról:

```
user.postValue(User("Zoe", 24))
```

Javasolt architektúra

- Az *Activity*-k, *Fragmentek*, egyedi nézetek *ViewModel*-eket használnak
- A *ViewModel*-ek *LiveData*-kon keresztül teszik megfigyelhetővé az adatokat
- A *ViewModel*-ek *Repository*-kat használnak az adatforrások elrejtéséhez
- Perzisztencia használata offline működés támogatására



Tanácsok

- A komponensek nem adatforrások
- Számos best practice létezik már, pl. MVVI, MVP, amik már beváltak
- Ha van egy bevált architektúránk, nem kell változtatni
 - > De érdemes megnézni, hol segíthetnek/egyszerűsíthetnek az architektúra komponensek
- Tervezéskor a tesztelhetőség mindig fontos szempont legyen



Architektúra referencia megvalósítások

- Útmutatók:
 - > <https://medium.com/androiddevelopers/viewmodels-a-simple-example-ed5ac416317e>
 - > <https://www.raywenderlich.com/817602-mvi-architecture-for-android-tutorial-getting-started>
- Példa minta projektek Architecture Components, architektúra és legjobb library-k használatára:
 - > <https://github.com/PatilShreyas/Foodium>
 - > <https://github.com/android/sunflower/tree/master/app>
 - > <https://github.com/Eli-Fox/LEGO-Catalog>

Architektúrák vizsgálata

Miért van szükség architektúrákra?

- Clean Code elvek
- Activity, Fragment osztályok könnyen túl nagyra nőhetnek
 - > Életciklus + business logic specifikus kód
 - > Nehéz tesztelni
- Valami mentén bontsuk rétegekre az alkalmazást

MVC, MVP, MVVM

- **MVC:**
Model View Controller
- **MVP:**
Model View Presenter
- **MVVM:**
Model View ViewModel

Rainbowcake architektúra

<https://rainbowcake.dev/>

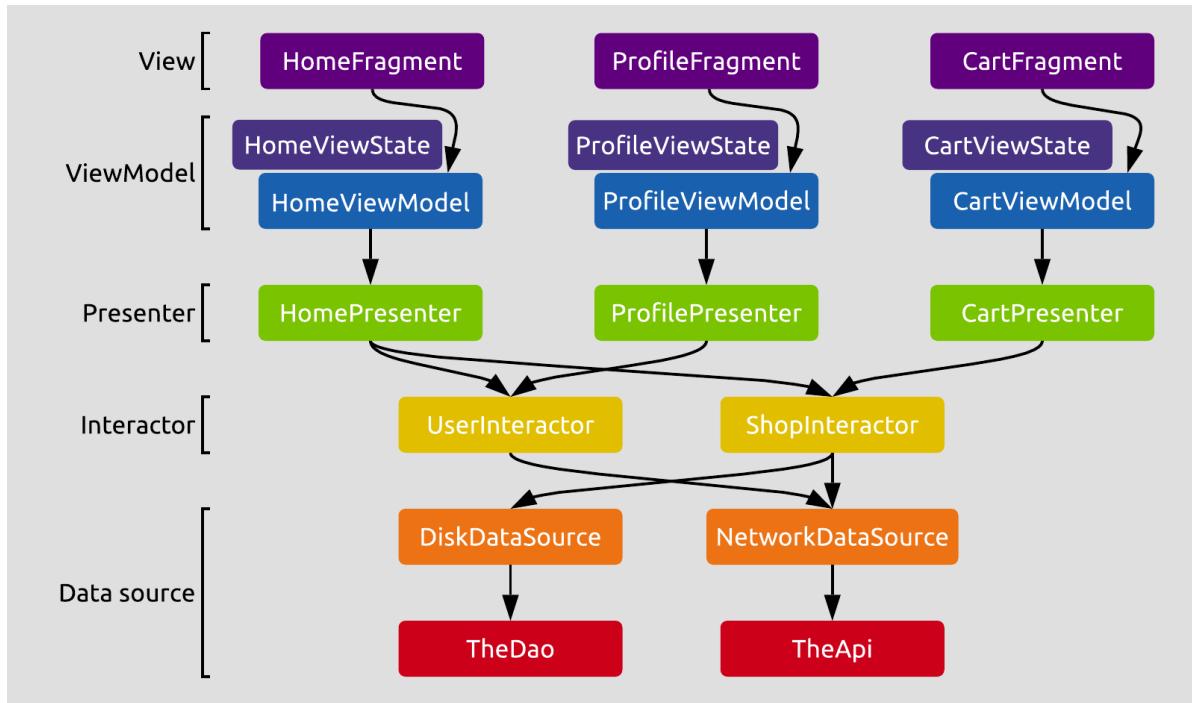
Braun Márton



Az architektúra fő céljai

- Rétegek és komponensek egyértelmű elválasztása
- A nézetek folyamatosan biztonságos és konzisztens állapotban vannak
- Konfiguráció változások megfelelő kezelése
- Könnyűvé tenni hosszabb ideig tartó műveletek háttér szálón való végrehajtását

Az architektúra elemei



Az architektúra elemei

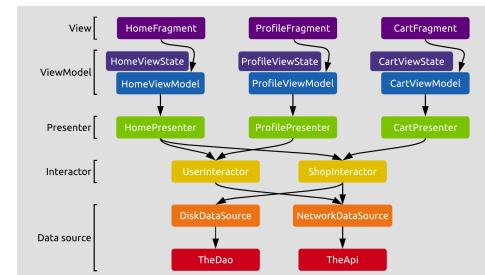
- **Views** (Fragments or Activities):
 - > Alkalmazás képernyők
 - > A ViewModel-ek állapotát figyelik és a megjelenítésért felelősek
 - > Eseményeket továbbítanak a ViewModel-nek
 - > Állapot változásokról, eseményekről értesülhetnek
- **ViewModel:**
 - > UI állapotát tárolja
 - > Kezeli a UI-al kapcsolatos logikát
 - > Frissíti az állapotot a presentertől kapott eredmények alapján
 - > Coroutine-okat indítanak minden input eseménye hatására és a presenter felé továbbítják a kérést
- **Presenter:**
 - > Háttér szálra helyezi a végrehajtást
 - > Interactor(oka)t használnak az üzleti logika elérésére
 - > Transzformálják az eredményt a képernyő számára szükséges formátumra (ami kell a ViewModel ViewState-jének)

- **Interactor:**

- Alsó szintű üzleti logikát tartalmazza
- Adat manipuláció, aggregálás, számítások végzése
- Nincsenek egy képernyőhöz kötve

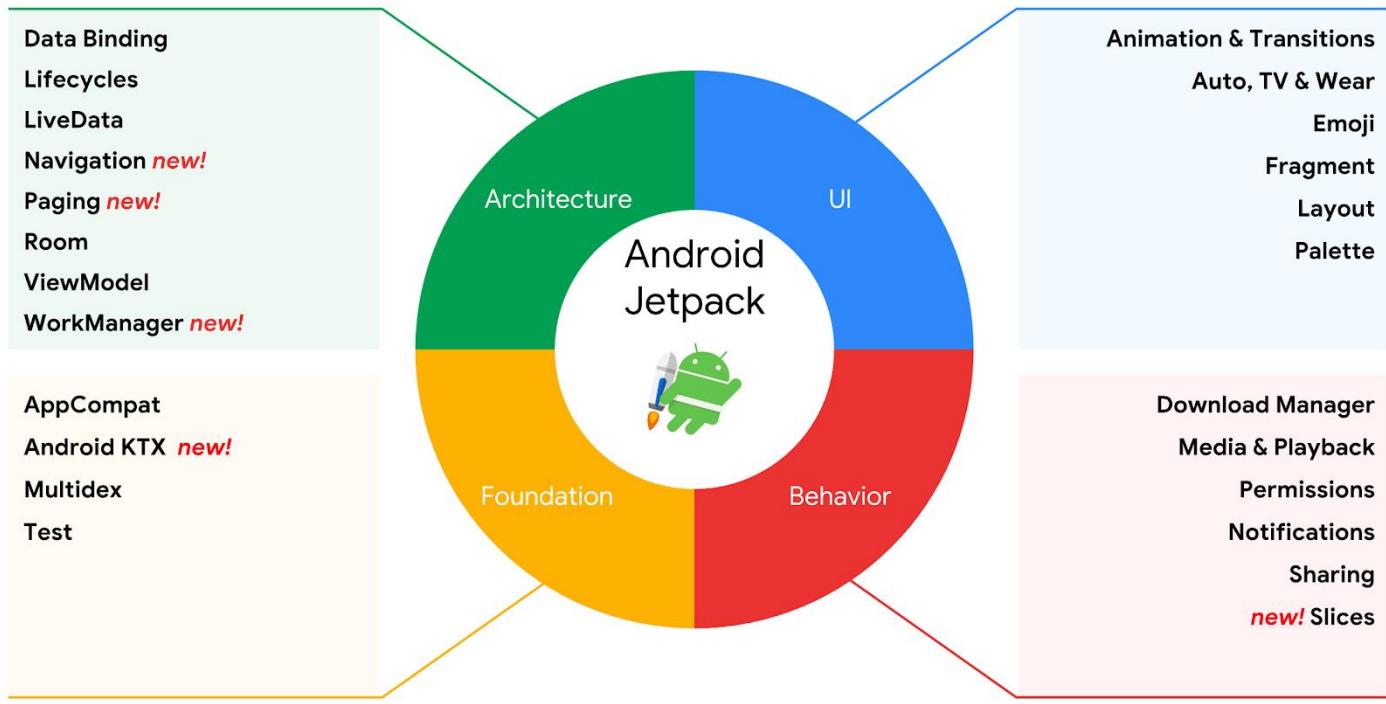
- **Data Source:**

- Interactorokat látja el adatokkal (adatbázis, file rendszer, hálózat, stb.)
- Implementáció és domain réteg szétválasztása
- Adatok konzisztens tárolása



Android Jetpack

Jetpack építő elemek



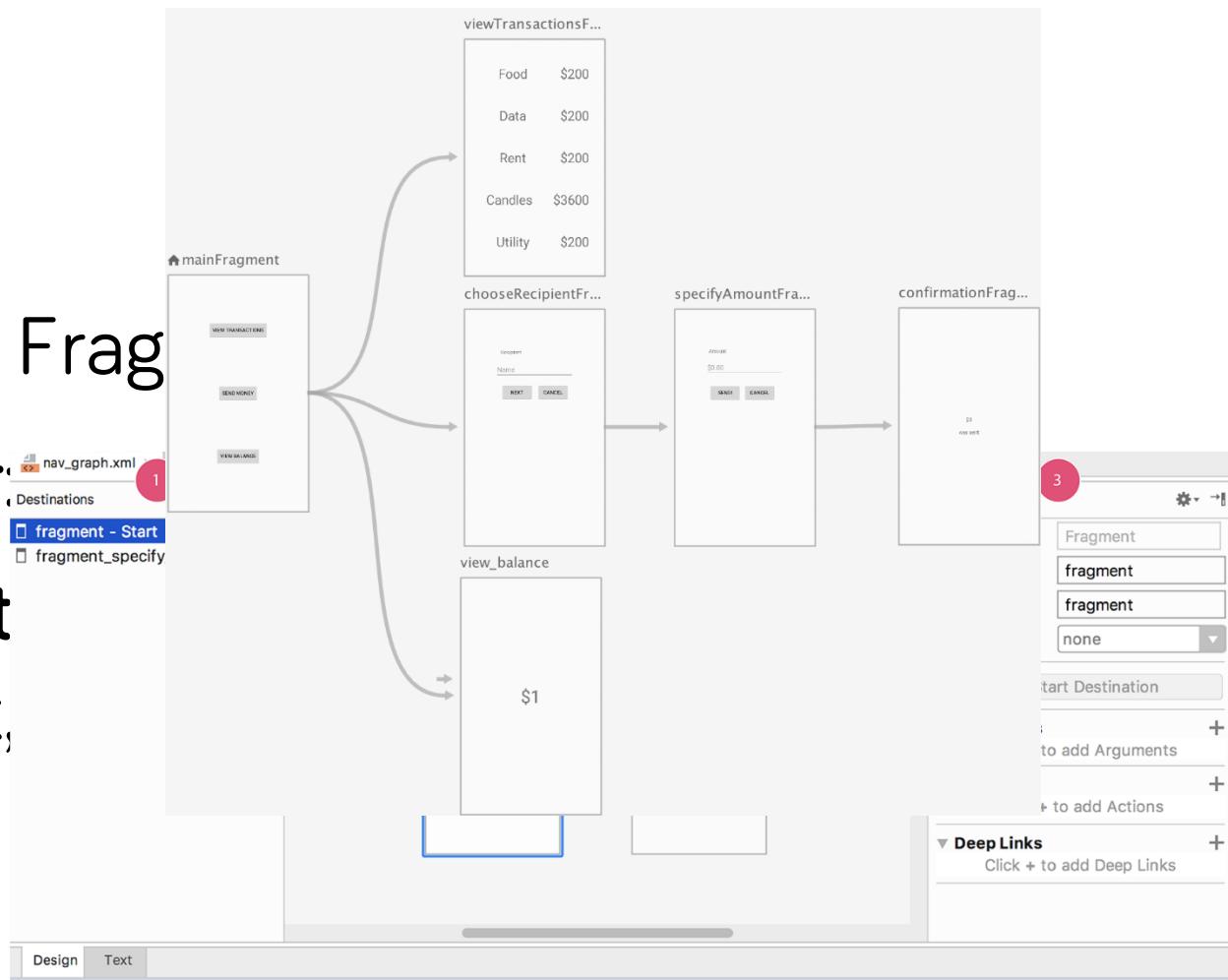
Forrás: <https://android-developers.googleblog.com/2018/05/use-android-jetpack-to-accelerate-your.html>

Navigáció

1. Célok listája: Frag

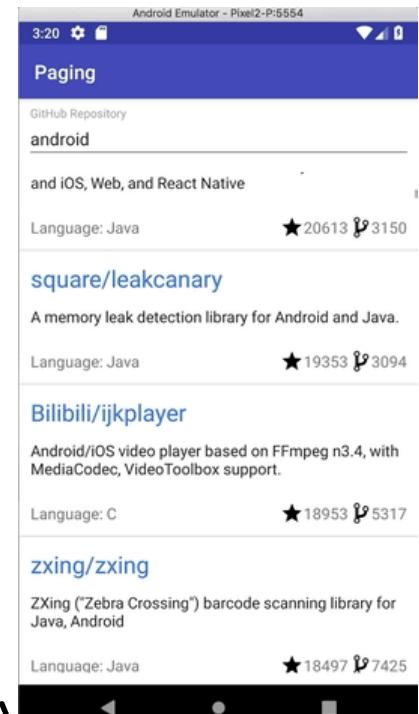
2. Graph Editor

3. Attribute edit
paraméterek,



Lapozás (Paging)

- Nagy mennyiségű adatok szakaszos betöltése
- Hálózati forgalom optimalizálása
- Performancia növelése
- Erőforrás használat minimalizálása
- Közvetlen támogatás:
 - > Room, LiveData, RxJava
- *PagedList* és *PagedListAdapter* osztályok



WorkManager

- Feladat irányító és ütemező szolgáltatás
- Kényszerek (*Constraint*) definiálása feladatok végrehajtására
 - > Töltés folyamatban
 - > Hálózati kapcsolat elérhető (pl. WiFi)
 - > Stb.
- A feladat lefut amint a **körülmények kedvezőek**
 - > A hívó komponensnek nem kell már előtérben lennie
- **Háttér szálon** indul a feladat
 - > Worker leszármazott objektum *doWork(...)* függvénye
- Végrehajtási mód **igazodik** az aktuális Android verzióhoz



Worker példa

```
class MyWorker(appContext: Context, workerParams: WorkerParameters) : Worker(appContext, workerParams) {

    override fun doWork(): Result {
        val handlerMain = Handler(Looper.getMainLooper())

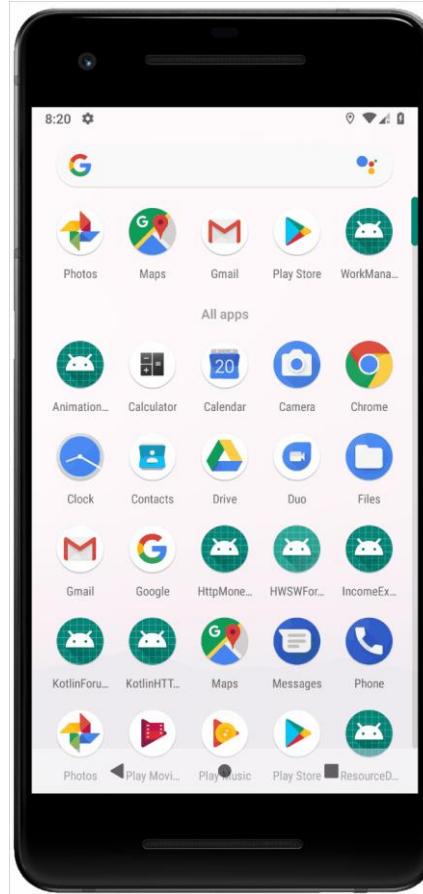
        handlerMain.post {
            Toast.makeText(applicationContext, "Hello", Toast.LENGTH_LONG).show()
        }

        return Result.SUCCESS;
    }
}

val constraints = Constraints.Builder()
    .setRequiresCharging(true)
    .setRequiredNetworkType(NetworkType.CONNECTED)
    .build()
val workRequest = OneTimeWorkRequest.Builder(MyWorker::class.java).setConstraints(constraints).build()
WorkManager.getInstance().enqueue(workRequest)
```

Worker példa

- Worker aktiváció
hálózat kapcsolódást
követően



Architecture Components – használjuk?

libraries > Android Architecture Components

Last updated: November 20, 2018

Android Architecture Components



A collection of libraries that help you design robust, testable, and maintainable apps.

License Apache License 2.0

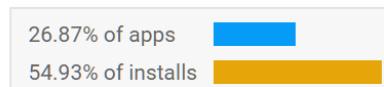
Tags #1 in App Frameworks, #2 in Database, #1 in Open Source

Website <https://developer.android.com/topic/libraries/architecture>



Statistics

Market share overall



Market share in top apps



Market share in new apps



Top apps that contain Android Architecture Components

Samsung Gallery
Samsung Electronics Co., Ltd.
★ 4.6 | Free | 100,000,000+

Google Play services
Google LLC
★ 4.0 | Free | 5,000,000,000+

Themes
Huawei Technologies Co., Ltd.
★ 4.0 | Free | 100,000,000+

Mi Video
Xiaomi Inc.
★ 4.0 | Free | 100,000,000+

WhatsApp Messenger
WhatsApp Inc.
★ 4.4 | Free | 1,000,000,000+

Samsung Experience Service
Samsung Electronics Co., Ltd.
★ 3.5 | Free | 100,000,000+

Forrás: https://www.appbrain.com/stats/libraries/details/android_arch/android-architecture-components

Android KTX

- Kotlin még hatékonyabb használata Androidon
- Android API-k kiegészítése, optimalizálása
- Rövidebb, olvashatóbb kód
- Fő elemek:
 - > Core KTX: alap API-k: Toast, Span, Menu
 - > Fragment KTX: Fragment tranzakciók egyszerűsítése
 - > Palette KTX: Color palette API egyszerűbb használata
 - > SQLite KTX: SQLite műveletek egyszerűsítése
 - > Collections KTX: Collection API egyszerűsítés
 - > Navigation KTX: Jetpack Navigation használata



Android KTX példák

```
sharedPreferences.edit()  
    .putBoolean("key", value)  
    .apply()
```



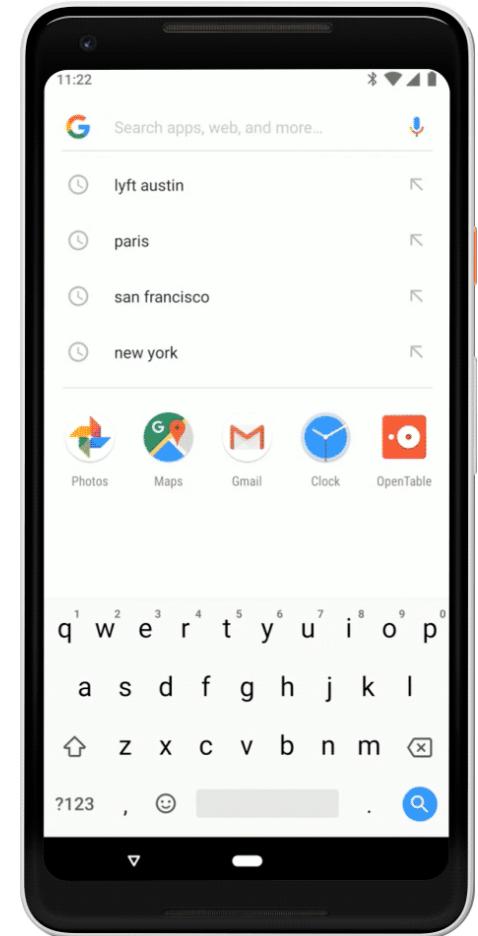
```
sharedPreferences.edit {  
    putBoolean(key, value)}
```

```
val drawable = bitmap.toDrawable(resources)
```

```
Month.APRIL.asInt()  
Year.now().asInt()
```

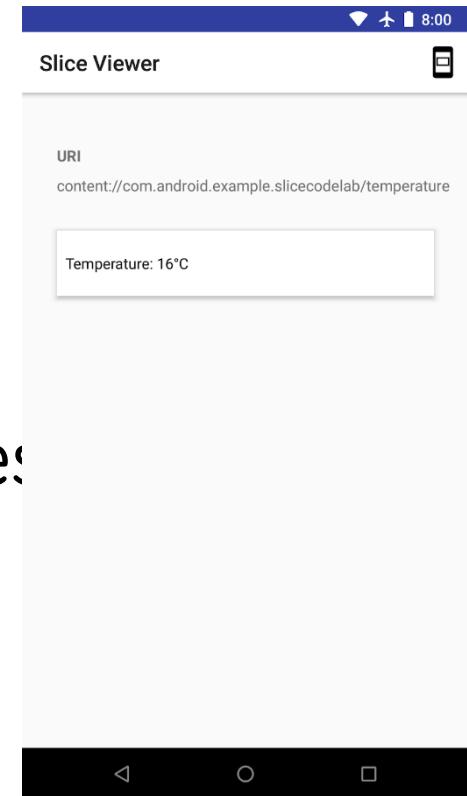
Slices

- Felhasználói élménye növelés
- Információ és fő funkciók megjelenítése azonnal
 - > Google kereső
 - > Google Assistant
- Statikus és interaktív tartalmak megjelenítése
- Közvetlen kapcsolat a teljes alkalmazással

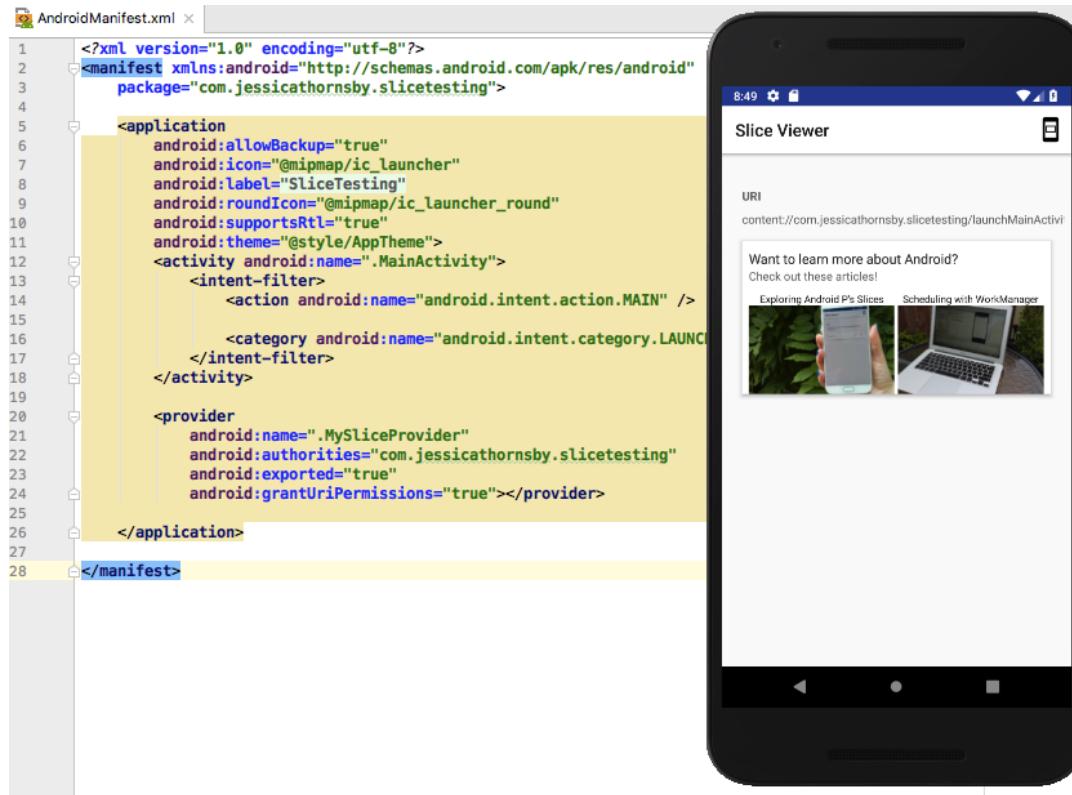


Slices példa

- ContentProvider regisztráció
 - > Manifest: <provider>...</provider>
- Saját *SliceProvider* osztály
 - > Slice nézet elkészítése
- Interakció BroadcastReceiver-en keresztül
- Tesztelés Slice Viewer-el



Slices példa



forrás: <https://www.androidauthority.com/android-jetpack-android-support-library-878587/>

Változó az alkalmazás viselkedési modellek

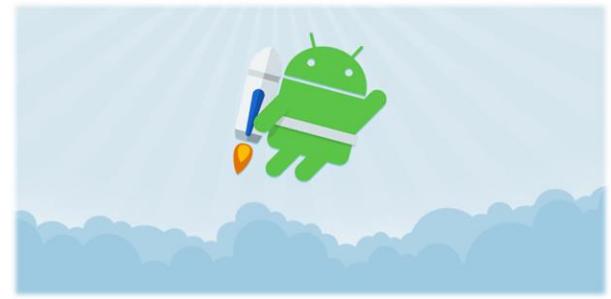


AndroidX

- **Support Library:** cél a kompatibilitás fenntartása (amennyiben lehetséges)
- **JetPack:** közös irányelvek, eszközök, módszertanok és egységes könyvtárak
- **AndroidX:** kompatibilitás támogatás hosszú távon

```
android.support.** > androidx.@  
android.databinding.** > androidx.databinding.@  
android.design.** > com.google.android.material.@  
android.support.test.** > androidx.test.@
```

Összegzés



- Megéri megismerni a Jetpacket?
 - > Közös architektúra
 - > Ismert kód struktúra
 - > Közös fejlesztési elvek és módszertanok
- Érdekességek
 - > Mi nem része a Jetpacknek?
 - > Változó alkalmazás viselkedési formák
 - „One click business”

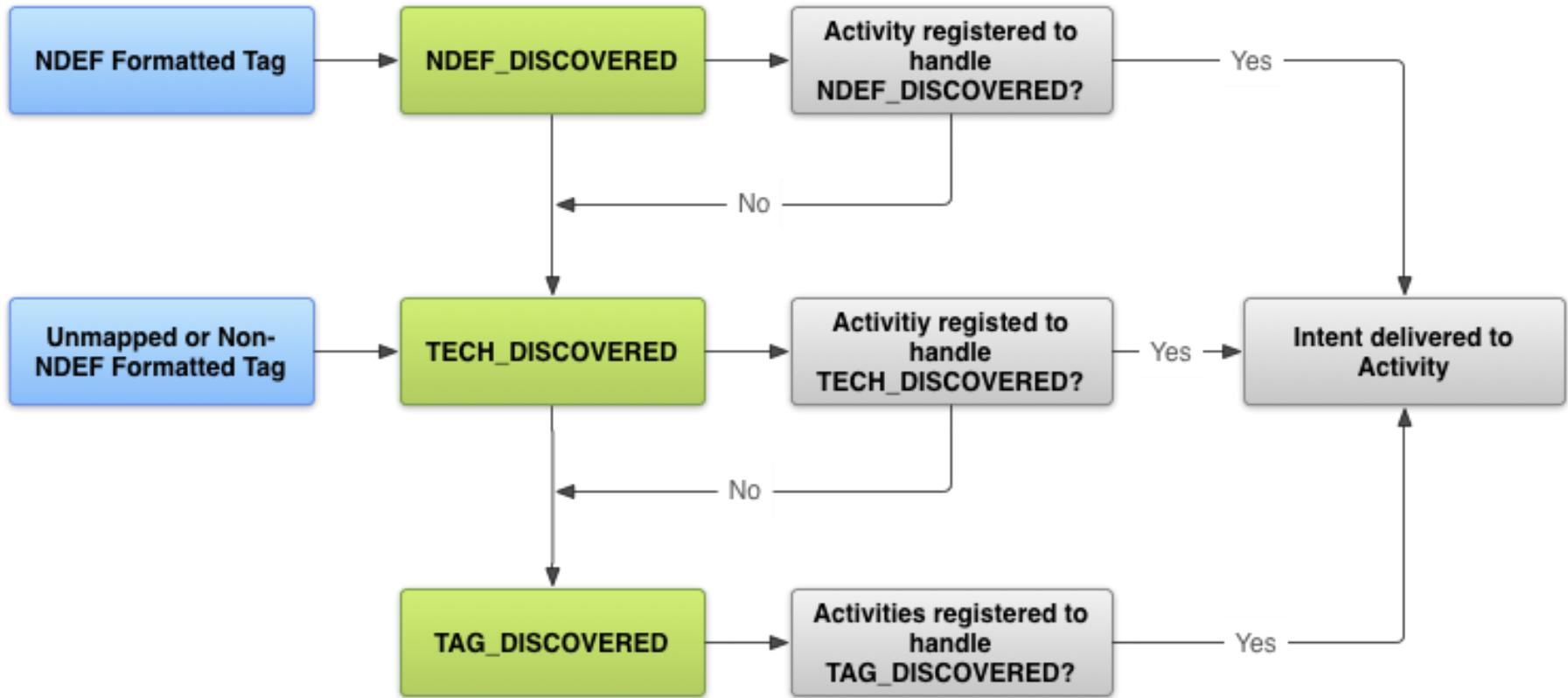
NFC

Near Field Communication

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)

Tag Dispatch System működése



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" />

Szöveges tartalom detektálása

- Manifest-en belül a megfelelő `<activity>`-elembe:

```
<intent-filter>  
    <action android:name="android.nfc.action.NDEF_DISCOVERED"/>  
    <category android:name="android.intent.category.DEFAULT"/>  
    <data android:mimeType="text/plain" />  
</intent-filter>
```

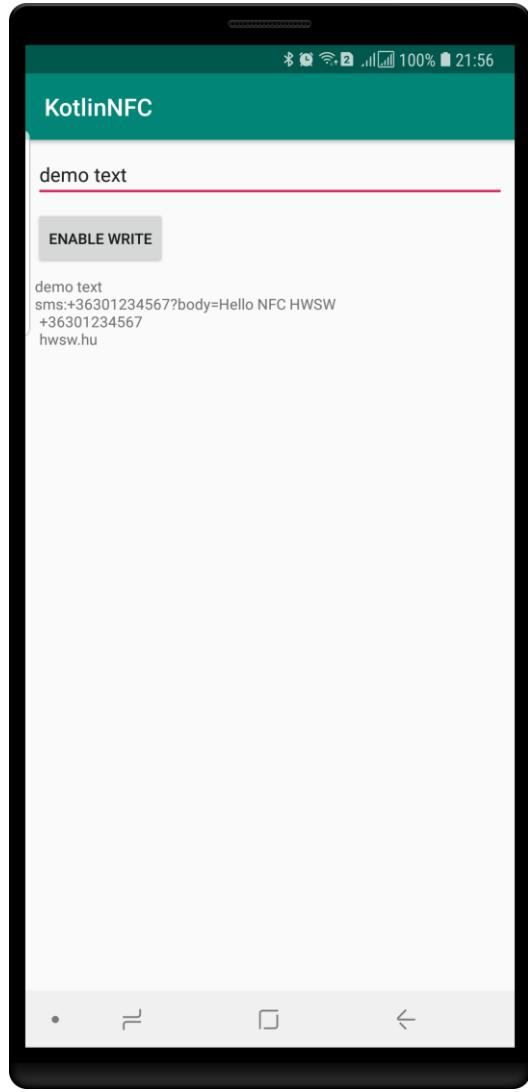
URI detektálása

- Szűrés a <http://www.aut.bme.hu/> URI-ra

```
<intent-filter>  
    <action android:name="android.nfc.action.NDEF_DISCOVERED"/>  
    <category android:name="android.intent.category.DEFAULT"/>  
    <data android:scheme="http" android:host="www.aut.bme.hu"  
        android:pathPrefix="/" />  
</intent-filter>
```

Első NdefRecord payload-jának kiolvasása

```
override fun onResume() {
    super.onResume()
    if (intent.action == NfcAdapter.ACTION_NDEF_DISCOVERED) {
        val parcelableArray = intent.getParcelableArrayExtra(
            NfcAdapter.EXTRA_NDEF_MESSAGES)
        parcelableArray?.forEach {
            val ndefMsg = it as NdefMessage
            ndefMsg?.records?.forEach {
                tvStatus.append("String(it.payload) \n")
            }
        }
    }
}
```



Gyakoroljunk!

- Készítsünk NFC író/olvasó alkalmazást
- Szöveges tartalom esetén induljon el automatikusan az Activity

Nearby API

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/>



Gyakoroljunk!

- Készítsünk egyszerű chat alkalmazást Nearby Messages API-val!

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

- Fő lépések:
 - > ACCESS_FINE_LOCATION engedély szükséges

- Feliratkozás:

```
Nearby.getMessagesClient(this).subscribe(messageListener)
```

- Leiratkozás:

```
Nearby.getMessagesClient(this).unsubscribe(messageListener)
```

- Üzenet küldése:

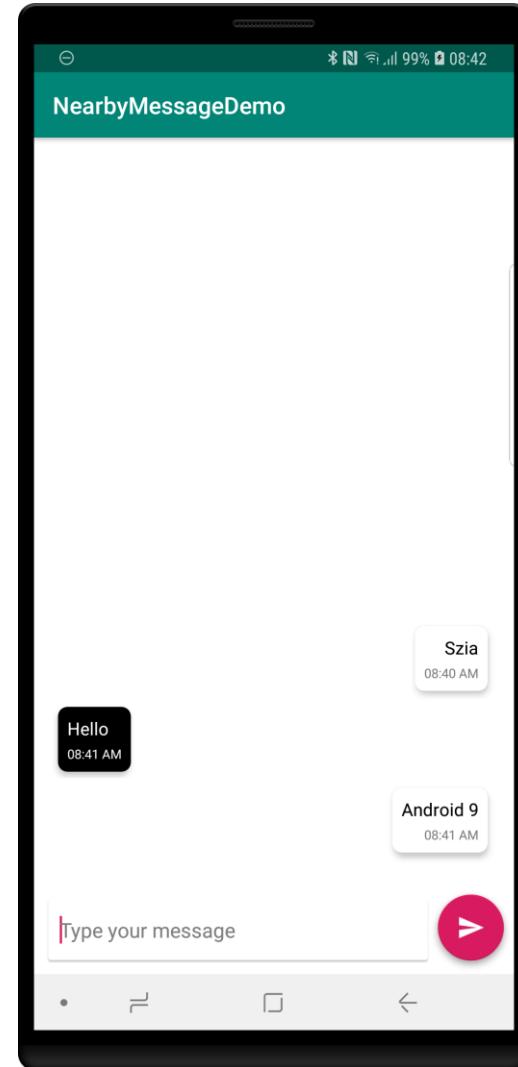
```
val nearByMessage = Message(message.toByteArray())
Nearby.getMessagesClient(this).publish(nearByMessage)
```

MessageListener megvalósítása

```
private var messageListener = object : MessageListener() {
    override fun onFound(
        message: com.google.android.gms.nearby.messages.Message) {
        // üzenet mint String: String(message.content)
    }

    override fun onLost(
        message: com.google.android.gms.nearby.messages.Message) {
        Toast.makeText(this@MainActivity,
            "LOST ${String(message.content)}",
            Toast.LENGTH_LONG).show()
    }
}
```

- Próbáljuk ki az Android Chat UI library-t
- <https://github.com/timigod/android-chat-ui>



Gyakran használt külső osztálykönyvtárak

Tippek & trükkök

- <https://github.com/nisrulz/android-tips-tricks>
- Eszközök, shortcutok
- Kódolási javaslatok
- UI/UX megoldások
- Kotlin ötletek
- Osztálykönyvtárak

Libek

- Retrofit
- RetroLambda
- Picasso
- ButterKnife
- Parceler
- IcePick
- LeakCanary
- Espresso
- Robolectric
- Dagger 2

Libek

- Zxing: QR kód olvasás
- Glide: képek betöltése hatékonyan
- MPAndroidChart
- OSM Droid: open street maps
- Google Protocol Buffers
- RxJava
- EventBus
- Android Annotations

Libek

- Apache Commons
- OpenCV
- Vuforia
- Tesseract OCR
- KSOAP2
- Oauth-signpost
- ...

Játék motorok

- libGDX: <http://code.google.com/p/libgdx/>
- min3d: <http://code.google.com/p/min3d/>
- Ogre3D:
<http://www.ogre3d.org/tikiwiki/Ogre+Android>
- Unity:
<http://unity3d.com/unity/multiplatform/mobile>
- Box2D: <http://code.google.com/p/androidbox2d/>
- Cocos2D Android port:
<http://code.google.com/p/cocos2d-android/>

Lib collections

- <http://blog.autsoft.hu/the-android-and-ios-alternative-library-collection/>
- <https://github.com/wasabeef/awesome-android-ui/blob/master/README.md>
- https://github.com/codepath/android_guides/wiki/Must-Have-Libraries
- <https://android-arsenal.com/>
- ...

További eszközök

- Grafika/ikonok
 - > <https://romannurik.github.io/AndroidAssetStudio/icons-launcher.html>
 - > <https://unsplash.com/>
 - > <https://github.com/nickbutcher/plaid>
- Android Studio Pluginek
 - > <https://plugins.jetbrains.com/plugin/7275-codeglance>
 - > <https://plugins.jetbrains.com/plugin/7495--ignore>
 - > <https://plugins.jetbrains.com/plugin/8527-google-java-format>
 - > <https://plugins.jetbrains.com/plugin/8533-json2pojo>

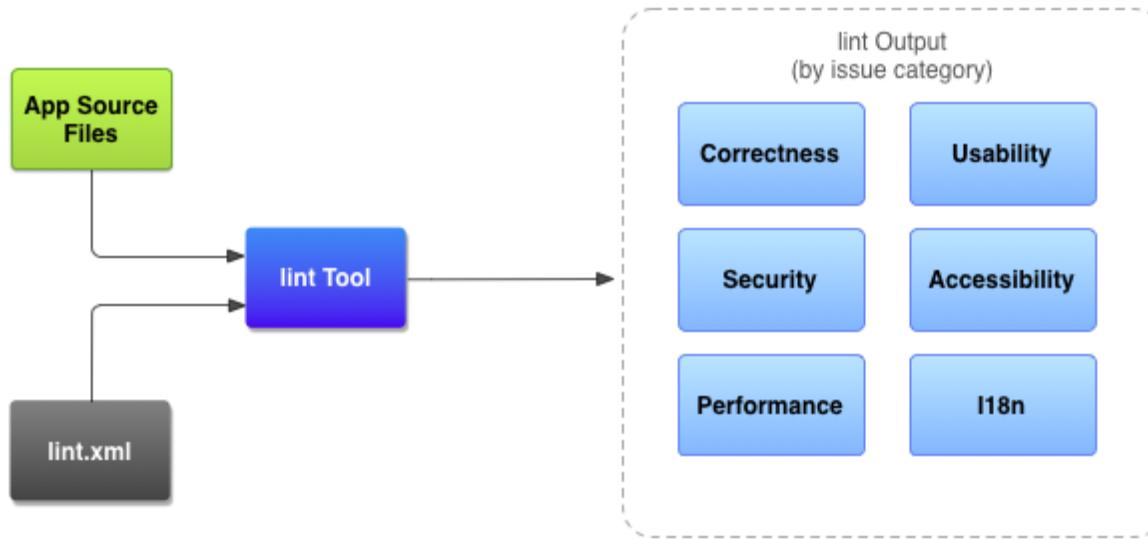
Statikus kódelemzés

Lint eszköz - kódvizsgálat

- Bekapcsolás:
 - > Window > Show View > Other > Android > Lint Warnings
- Kód strukturális problémák felderítése
- Hatékonyság, teljesítmény növelése (például nem használt XML namespacek kiszűrése)
- Deprecated kódrészek feltárása és javaslat tétele

Lint konfigurálása

- Konfiguráció a lint.xml-en keresztül (beállítható, hogy mi kerüljön ellenőrzésre), valamint @SuppressLint
 - > lint.xml a projekt gyökér könyvtárában helyezhető el
- Különböző figyelmeztetés és hiba szintek
- Parancssorból is vezérelhető, így például beilleszthető Continuous Integration környezetbe



lint.xml példa

```
<?xml version="1.0" encoding="UTF-8"?>
<lint>
    <!-- Disable the given check in this project -->
    <issue id="IconMissingDensityFolder" severity="ignore" />

    <!-- Ignore the ObsoleteLayoutParams issue in the specified
files -->
    <issue id="ObsoleteLayoutParams">
        <ignore path="res/layout/activation.xml" />
        <ignore path="res/layout-xlarge/activation.xml" />
    </issue>

    <!-- Ignore the UselessLeaf issue in the specified file -->
    <issue id="UselessLeaf">
        <ignore path="res/layout/main.xml" />
    </issue>

    <!-- Change the severity of hardcoded strings to "error" -->
    <issue id="HardcodedText" severity="error" />
</lint>
```

API érdekességek

Android API mérete

- Mit nevezünk API-nak?
- Mennyire ismerjük az API-t?
- Mennyire követjük az API változásokat?
 - > https://developer.android.com/sdk/api_diff/25/changes.html
- Mennyi osztályt/függvényt tartunk átlagosan fejben?
- Miről nem lesz szó?
 - > 3rd party osztálykönyvtárak

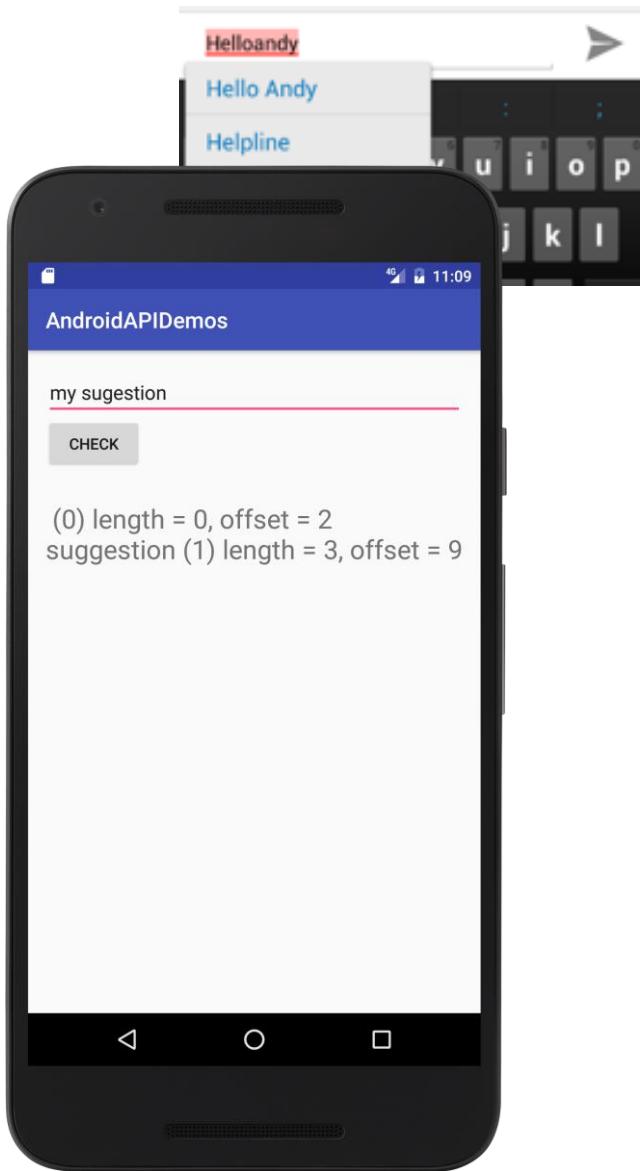
Java API

Java Development Kits	Codename	Release	Packages	Classes
Java SE 8 with JDK 1.8.0	---	2014	217	4,240
Java SE 7 with JDK 1.7.0	Dolphin	2011	209	4,024
Java SE 6 with JDK 1.6.0	Mustang	2006	203	3,793
Java 2 SE 5.0 with JDK 1.5.0	Tiger	2004	166	3,279
Java 2 SE with SDK 1.4.0	Merlin	2002	135	2,991
Java 2 SE with SDK 1.3	Kestrel	2000	76	1,842
Java 2 with SDK 1.2	Playground	1998	59	1,520
Development Kit 1.1	---	1997	23	504
Development Kit 1.0	Oak	1996	8	212

Forrás: [Java 8 Pocket Guide](#) book by Robert Liguori, Patricia Liguor

SpellChecker API

- Helyesírás ellenőrzése
- Szavak és akár mondatok ellenőrzése is
- Mondat ellenőrzés:
 - > `Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN`
- Spell Check Service készíthető az automatikus működéshez
 - > `TextView` leszármazottakat automatikusan tudja ellenőrizni
- Kérdés: Mikor tudjuk ezt használni?



TextRecognizer használati példa

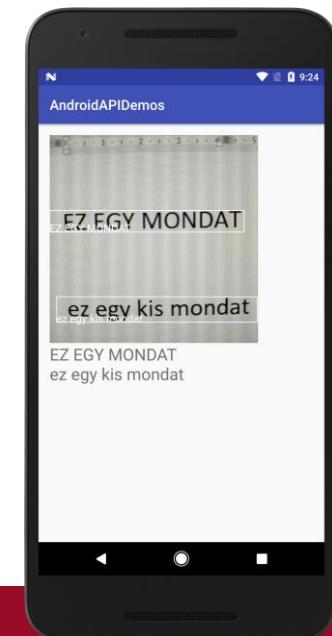
- Szöveg és mondat felismerés

```
TextRecognizer textRecognizer = new  
TextRecognizer.Builder(context).build();
```

- Szöveg blokkok azonosítása

```
public class OcrDetectorProcessor implements Detector.Processor<TextBlock> {  
  
    @Override  
    public void receiveDetections(Detector.Detections<TextBlock> detections) {  
        ...  
        SparseArray<TextBlock> items = detections.getDetectedItems();  
        ...  
    }  
  
    @Override  
    public void release() {  
    }  
}
```

- Javasolt beállítás:
 - > android:keepScreenOn="true"



TimingLogger

- Időbényeg alapú loggolás

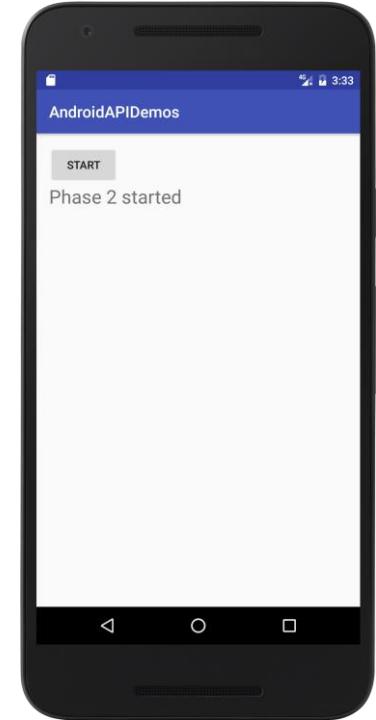
D/TAG_MYJOB: MyJob: begin

D/TAG_MYJOB: MyJob: 2002 ms, Phase 1 ready

D/TAG_MYJOB: MyJob: 2002 ms, Phase 2 ready

D/TAG_MYJOB: MyJob: 2001 ms, Phase 3 ready

D/TAG_MYJOB: MyJob: end, 6005 ms



- TimingLogger osztály ad támogatást
 - > Log.isLoggable(" TAG_MYJOB ", Log.VERBOSE); -
 - > false
- Engedélyezés:
 - > adb shell
 - > setprop log.tag.TAG_MYJOB VERBOSE

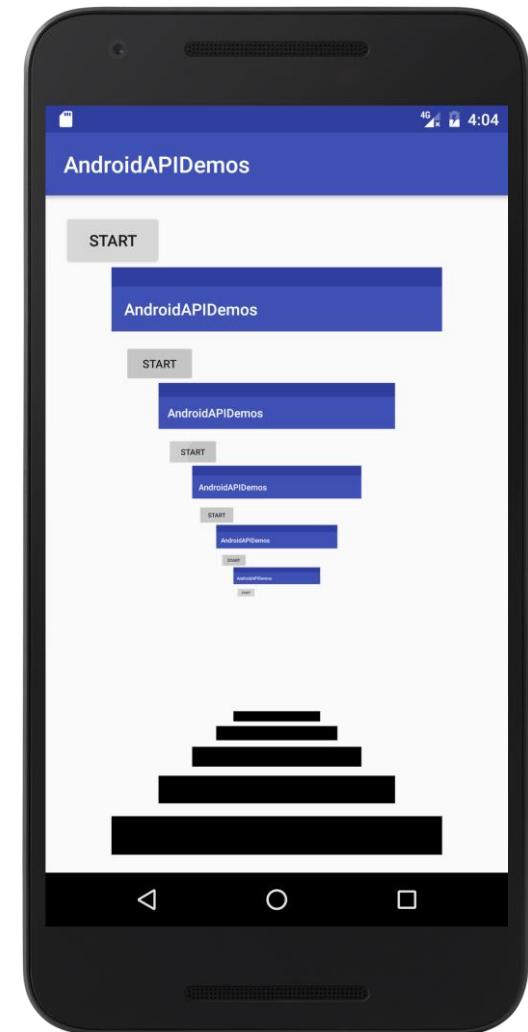
Képernyőkép mentés

- Képernyő képe egyszerűen elérhető:

```
View viewRoot =  
viewRoot.setDrawingCacheEnabled(true) ;  
Bitmap bitmap =  
Bitmap.createBitmap(viewRoot.getDrawingCache()) ;  
viewRoot.setDrawingCacheEnabled(false) ;
```

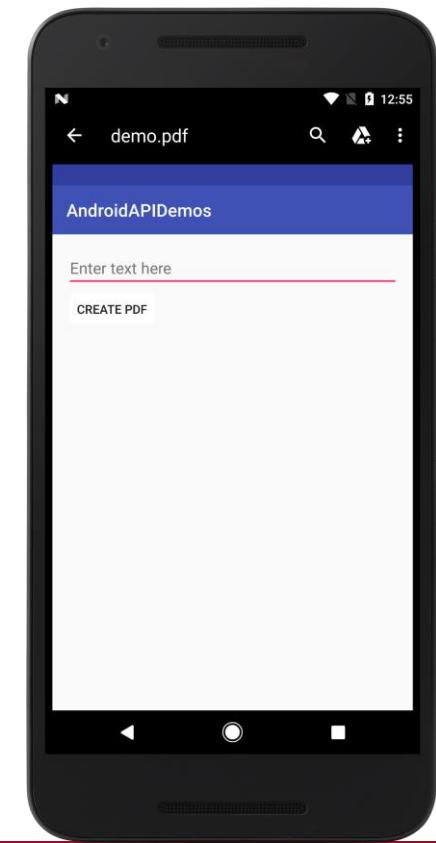
- Kérdés: Mikor lehet rá szükség?

- Real time: Media Projection API
 - > <https://github.com/googlesamples/android-ScreenCapture>



PDF készítés

- PDF dokumentum generálás
- Egy és több oldalas PDF dokumentumok támogatása
- Teljes értékű rajzolási lehetőség
- PDF dokumentum elmentése bármilyen *OutputStream*-be
- API 19-től érhető el
- Kérdés: Mikor lehet rá szükség?



PDF készítés folyamata

```
PdfDocument document = new PdfDocument();

 PageInfo pageInfo = new PageInfo.Builder(
    new Rect(0, 0, 100, 100), 1).create();

 Page page = document.startPage(pageInfo);

 View content = getContentView();

 content.draw(page.getCanvas());

 document.finishPage(page);

 document.writeTo([outputStream]);

 document.close();
```

Android 10-12



Android verziók

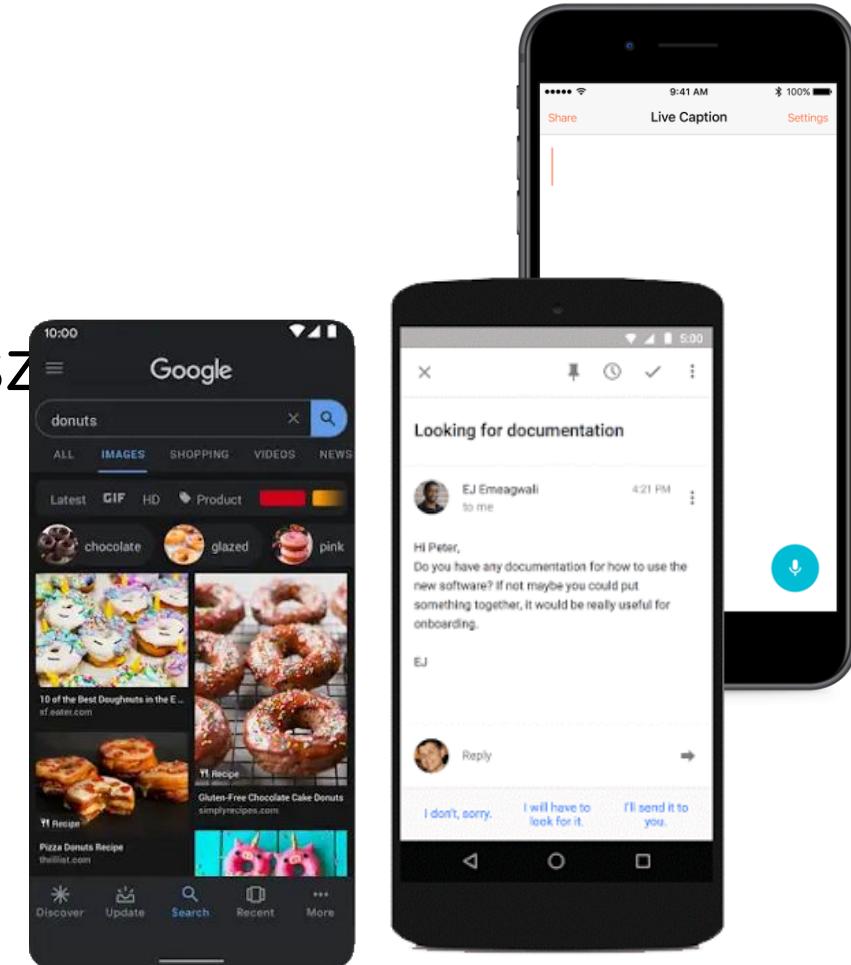
A B C D E F G H I J
K L M N O P 10 11
12 13 14 ...

Még van $\sim\infty$ évünk ☺



Android 10-12 - főbb újdonságok

- Élő feliratok
- Intelligens gyors válaszok
- Intelligens hangerő és zajszabályozás
- Újfajta gesztusok
- Sötét téma
- Biztonsági fejlesztések
- Focus mode
- Family Link



Hány százalékban változott az API?

- Pie -> 10? (%)
 - > **6.64%**

Type	Additions	Changes	Removals	Total
Packages	6	88	0	94
Classes and Interfaces	167	585	0	752
Constructors	36	216	0	252
Methods	793	1443	14	2250
Fields	539	503	12	1054
Total	1541	2835	26	4402



https://developer.android.com/sdk/api_diff/30/changes

Biztonsági frissítések

- Fejlettebb azonosítási módok
 - > Biometrikus azonosítási integráció
 - > Komplett API azonosítási dialógusokra
- Beágyazott DEX kód futtatása közvetlenül APK-ból
 - > A lokálisan fordított kód hamisítása elleni jobb védelem
 - > Vigyázat: sebesség romolhat

```
aaptOptions {  
    noCompress 'dex'  
}
```

- Publikusan elérhető Conscrypt API



Hálózati kapcsolatok kezelése

- Wi-Fi network connection API
- Wi-Fi network suggestion API
- Wi-Fi high-performance és low-latency módok
- Wi-Fi Direct connection API



```
val specifier = WifiNetworkSpecifier.Builder()  
    .setSsidPattern(PatternMatcher("hwsn", PatternMatcher.PATTERN_PREFIX))  
    .setBssidPattern(MacAddress.fromString("10:03:23:00:00:00"), MacAddress.fromString("ff:ff:ff:00:00:00"))  
    .build()  
  
val request = NetworkRequest.Builder()  
    .addTransportType(NetworkCapabilities.TRANSPORT_WIFI)  
    .removeCapability(NetworkCapabilities.NET_CAPABILITY_INTERNET)  
    .setNetworkSpecifier(specifier)  
    .build()
```



Telefonhívások

- Hívás minőség fejlesztése
 - > IP hívások minőségével kapcsolatos információk
- Hívás azonosítás és felderítés
 - > SPAM hívások szűrése
 - > READ_CALL_LOG engedély
- Hívás átirányítás API
 - > *CallRedirectionService* a NEW_OUTGOING_CALL broadcast helyett
 - > Lehetőség a hívások átirányítására például VoIP irányba



Multimédia újdonságok



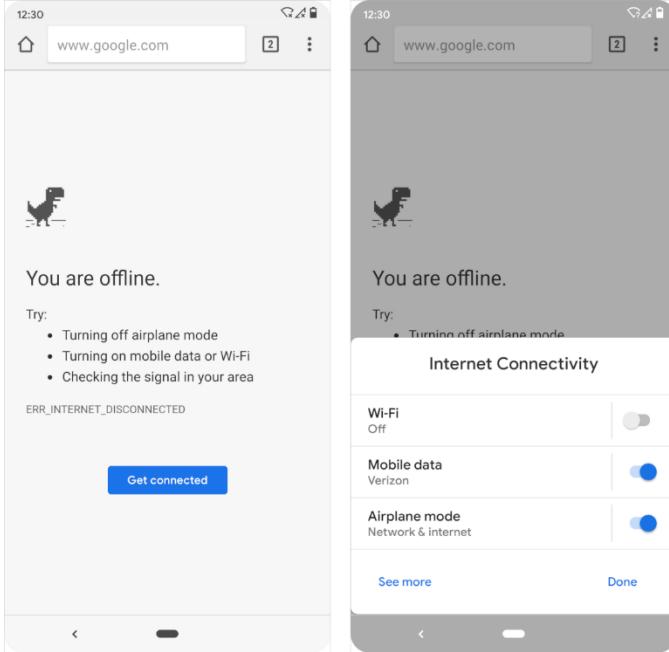
- Audio input megosztás
 - > Több alkalmazás megoszthatja a hangfelvételi csatornát
- Hanglejátszás megosztása alkalmazások között
 - > Például hallgatott zene streamelése mások számára
- Native MIDI API
 - > JNI overhead minimalizálása
- Media codec információk részletesebb elérése

Kamera és kép kezelés

- Monochrome kamera támogatás
- Dynamic Depth formátum
 - > Külön file-ba kerül eltárolásra a kép mellett
- High Efficiency Image File formátum (HIEC)
 - > Jobb minőség
 - > Kisebb file méret
- Több kamera egyidejű kezelése
 - > Camera2 API továbbfejlesztése



Egyszerűbb instant beállítások



```
val panellIntent = Intent(  
    Settings.Panel.settings_panel_type)  
startActivityForResult(panellIntent)
```

- ACTION_INTERNET_CONNECTIVITY
- ACTION_WIFI
- ACTION_NFC
- ACTION_VOLUME

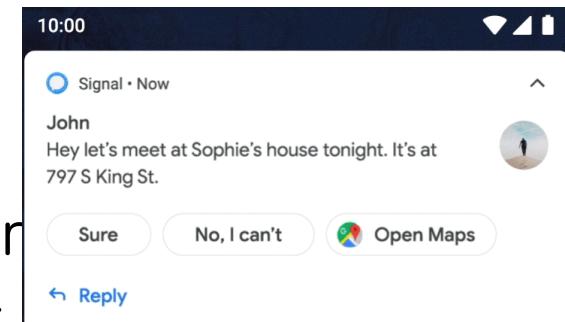
További érdekességek 1/2

- Thermal API
 - > Túlmelegedés esetén az alkalmazások kaphatnak értesítést és csökkenthetik a teljesítményt (pl. video stream minősége, stb.)
 - > HW módosítás is szükséges a készülékeken
- Accessibility továbbfejlesztése
- Autofill fejlesztések
- Sharing Shortcuts API



További érdekességek 2/2

- Foreground service típusok
 - > *connectedDevice*: például fitnesz jellegű készülékek integrációs szolgáltatása
 - > *dataSync*: nagyobb adat letöltése
 - > *location*
 - > *mediaPlayback*
 - > *mediaProjection*: készülék kiélezőjének felvétele
 - > *phoneCall*
- *TextClassifier*
 - > Nyelv felismerés
 - > Javasolt akciók szöveg kontextus alapjár
- Intelligens válaszok/akciók az értesítésekben

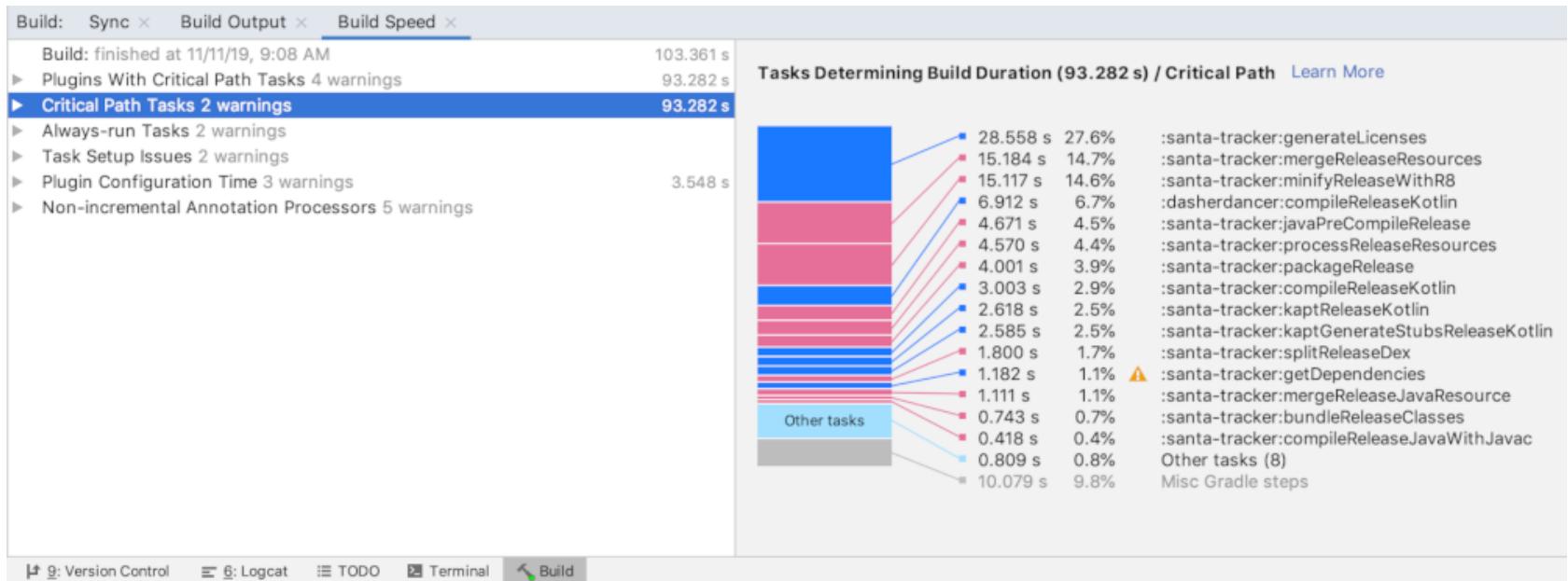


Android Studio újdonságok

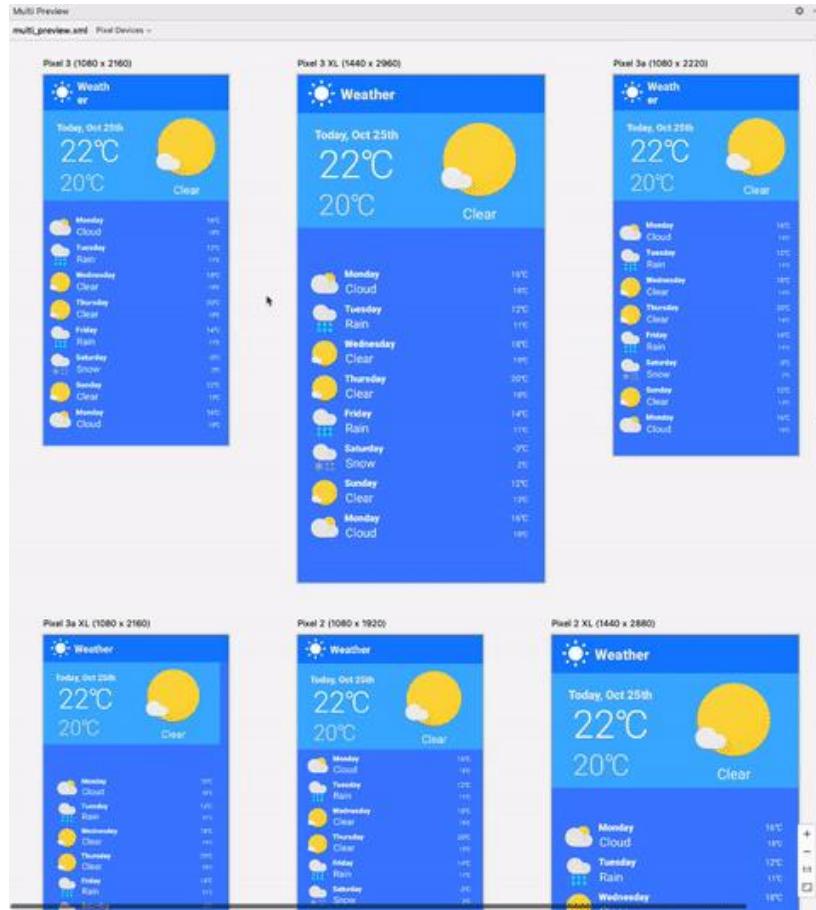
- 2013-tól érhető el
 - > Már 6 éves
- Android Studio 4.x – még csak canary csatornán
- Folyamatos fejlesztések és újdonságok
- Stabilitás és sebesség folyamatosan javul változik ☺



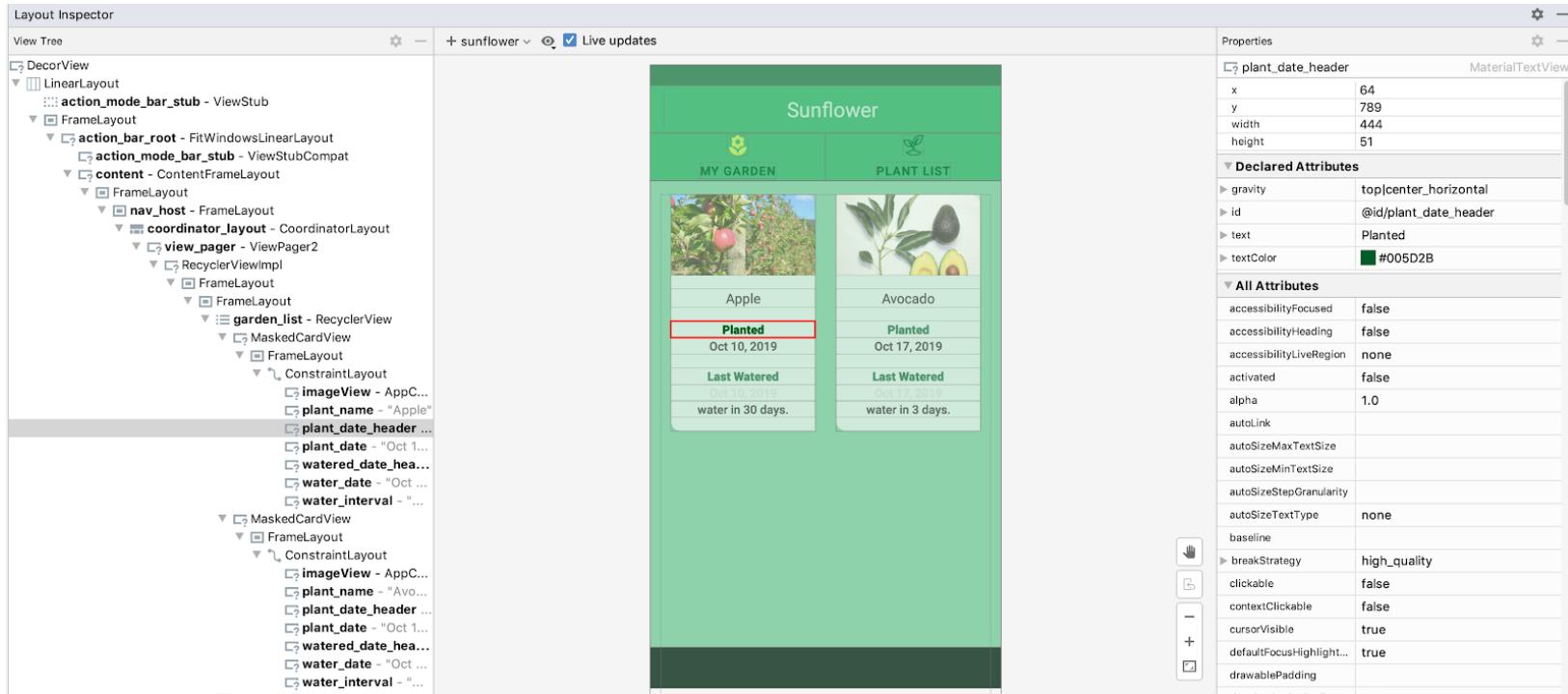
Fordítási idő nézet



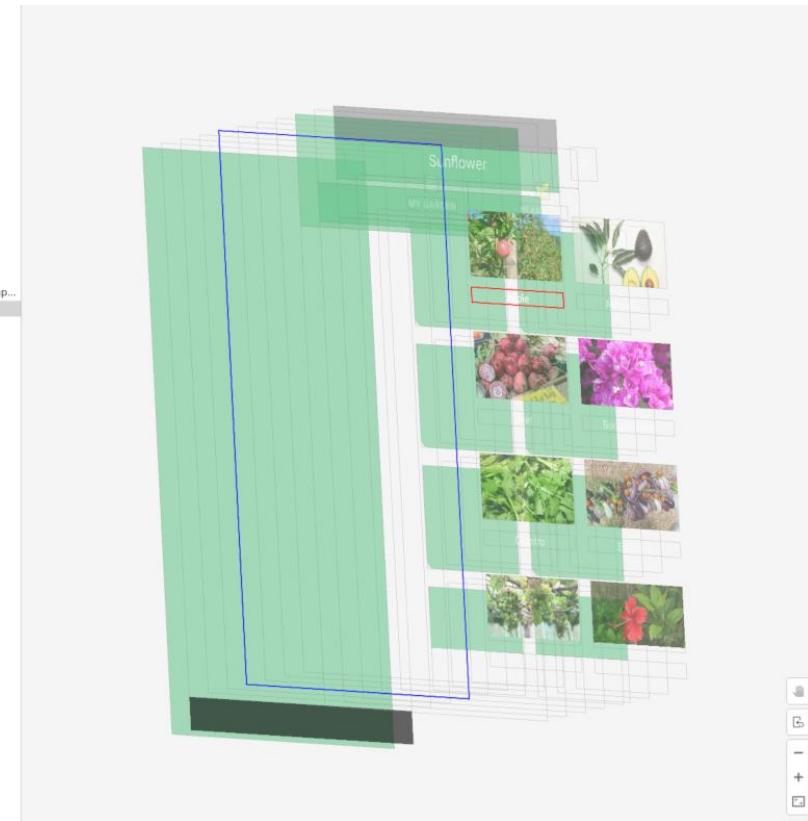
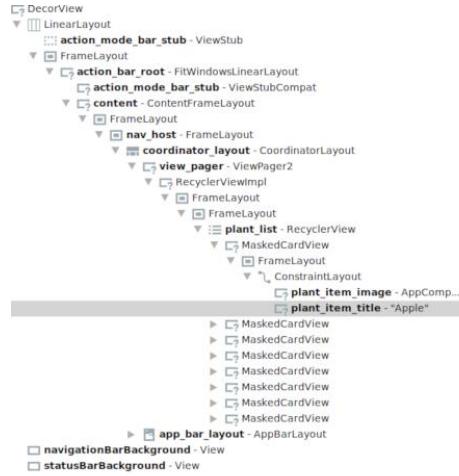
Multi preview



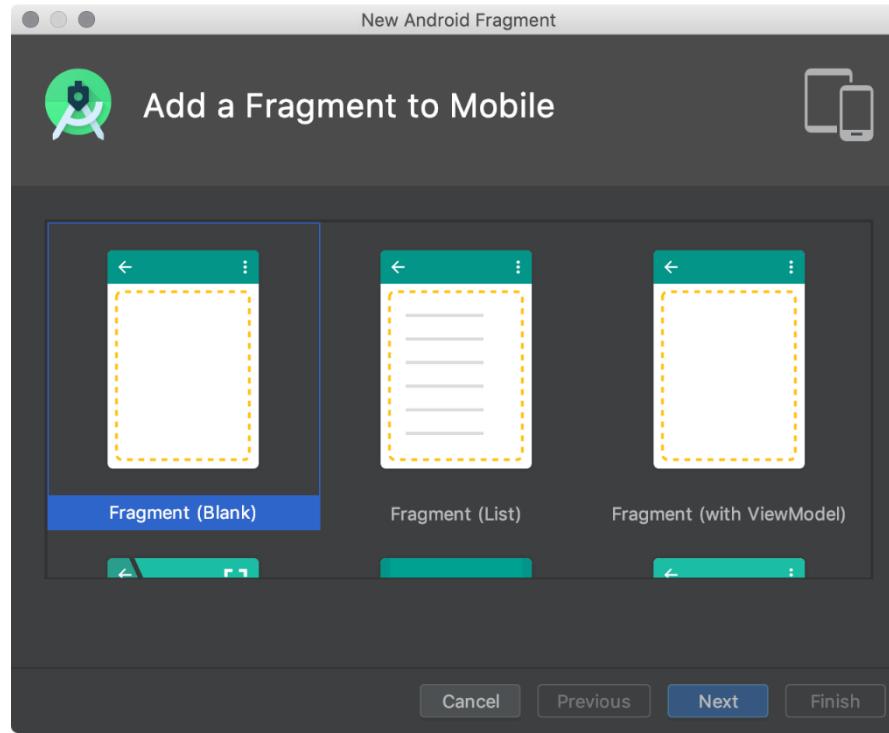
Live layout inspector



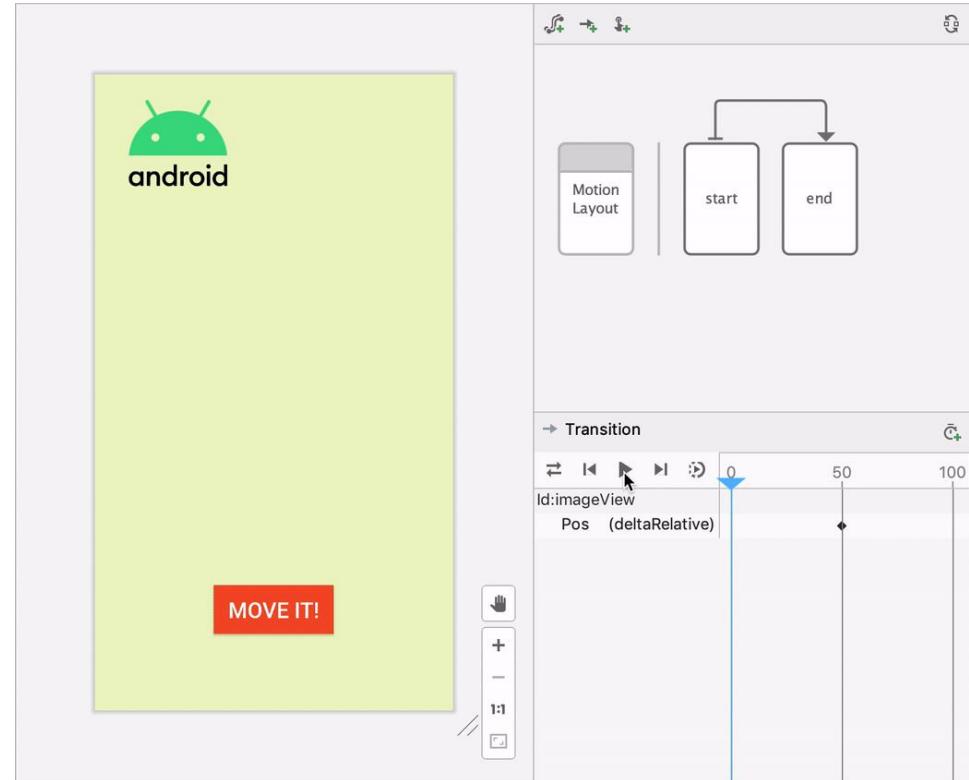
3D view



Fragment wizard



Motion Editor



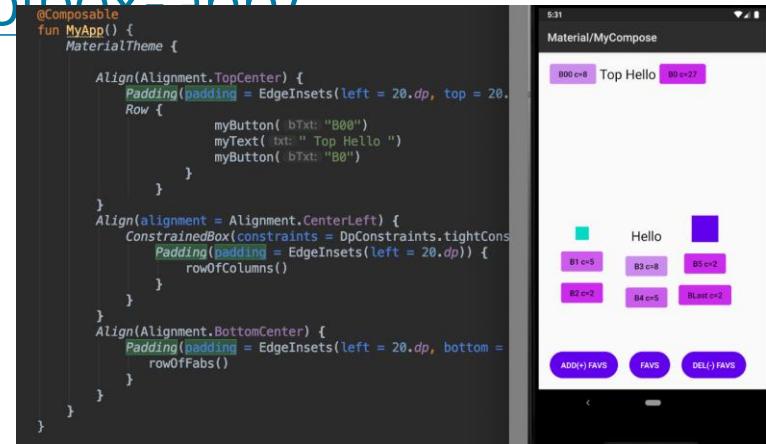
Jetpack compose

Jetpack compose

- Modern toolkit UI készítéshez
 - > Jelenleg beta verzió
 - (de telefonon fut már)
 - > A fejlesztő Kotlin kóddal leírja a UI paramétereit és a Compose motor generálja a felületet
 - > Könnyű a UI frissítése az alkalmazás állapota alapján
 - > Erőforrásokat (pl képek) ugyanúgy használhat
- Csomag része
 - > Eszközök
 - > Kotlin API-kat
- Előnyök
 - > Kevesebb kód
 - > Nincs szükség XML layout-ra
 - > Nincs szükség UI widgetek készítésére
 - > A UI elemek kódból készíthetők
 - > Könnyebb az újrafelhasználhatóság
 - > Kompatibilis a meglévő UI toolkit-el (XML – layout megoldással)

Előkövetelmények

- Legújabb Android Studio a Canary csatornáról
- Letöltési javaslat:
 - > JetBrains Toolbox
 - > <https://www.jetbrains.com/toolbox-app/>
- Compose eszközök
- Valós idejű preview
- Új projekt:
 - > *Empty Compose Activity template*



Composable függvények

- Segítségükkel készíthetők UI elemek Kotlin kódóból
 - > Alakzat és adat függőség megadása
- @Composeble annotáció a függvények elején
- Egymásba ágyazható UI elemek

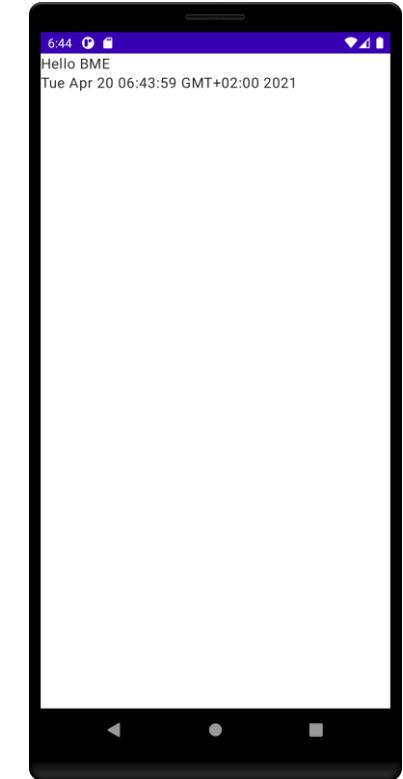
Jetpack Compose – HelloWorld

```
class DemoActivity : ComponentActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContent {  
            MaterialTheme {  
                HelloWorld()  
            }  
        }  
    }  
  
    @Composable  
    fun HelloWorld() {  
        Column {  
            Text("Hello BME")  
            Text(Date(System.currentTimeMillis()).toString())  
        }  
    }  
}
```

*ComponentActivit
y leszármazott*

*setContentView()
helyett setContent*

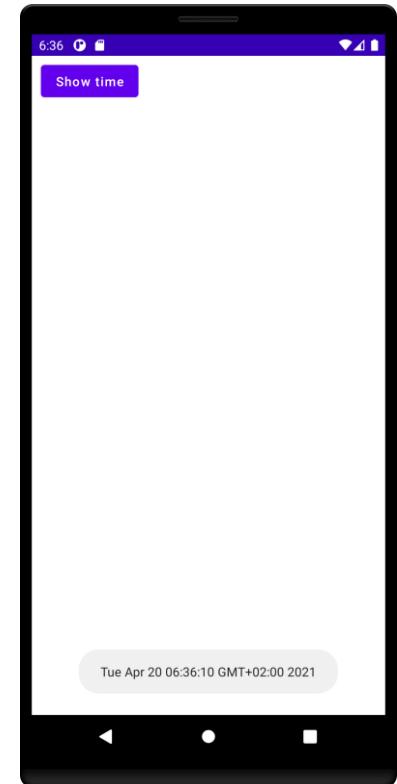
*@Composeable
függvény*



Jetpack Compose – Események kezelése

```
class DemoActivity : ComponentActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContent {  
            ButtonShowTime()  
        }  
    }  
  
    @Composable  
    fun ButtonShowTime() {  
        Button(  
            onClick = {  
                Toast.makeText(this, Date(System.currentTimeMillis()).toString(), Toast.LENGTH_LONG)  
                    .show()  
            },  
            modifier = Modifier.padding(Dp(10f))  
        ) {  
            Text("Show time")  
        }  
    }  
}
```

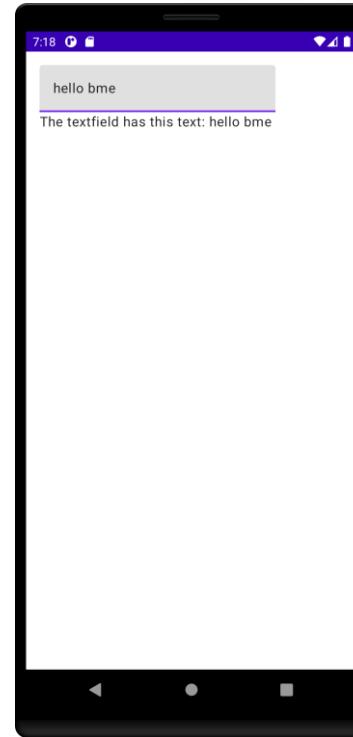
Eseménykezelő



Jetpack Compose – TextField (input)

```
class DemoActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MaterialTheme{
                TextFieldDemo()
            }
        }
    }

    @Composable
    fun TextFieldDemo() {
        Column(Modifier.padding(16.dp)) {
            val textState = remember { mutableStateOf(TextFieldValue()) }
            TextField(
                value = textState.value,
                onValueChange = { textState.value = it }
            )
            Text("The textfield has this text: " + textState.value.text)
        }
    }
}
```



Jetpack Compose – Fragment példa

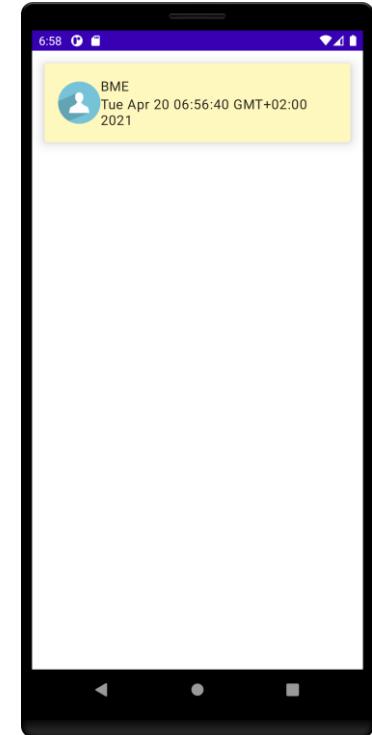
```
class ExampleFragment : Fragment() {

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        return ComposeView(requireContext()).apply {
            setContent {
                MaterialTheme {
                    Text("Hello BME Compose!")
                }
            }
        }
    }
}
```

Jetpack Compose – Képek és Card

```
@Composable
fun DemoCard(name: String) {
    Card(
        modifier = Modifier
            .fillMaxWidth()
            .padding(15.dp)
            .clickable {
                Toast.makeText(
                    this,
                    Date(System.currentTimeMillis()).toString(),
                    Toast.LENGTH_LONG
                ).show()
            },
        elevation = 10.dp,
        backgroundColor = Color(255,248,190)
    ) {
        Row(verticalAlignment = Alignment.CenterVertically, modifier = Modifier.padding(16.dp)) {
            Image(
                painter = painterResource(R.drawable.person),
                contentDescription = "Person",
                Modifier.size(50.dp, 50.dp),
                contentScale = ContentScale.Fit
            )
            Column {
                Text(name)
                Text(Date(System.currentTimeMillis()).toString())
            }
        }
    }
}
```

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
        MaterialTheme {
            DemoCard(name = "BME")
        }
    }
}
```



Jetpack Compose – listák kezelése

Adat
modell

```
data class Student(var name: String, var email: String)

@Composable
fun StudentsList(students: List<Student>) {
    LazyColumn() {
        items(students) {
            StudentCardSimple(it.name, it.email)
        }
    }
}
```

Lista

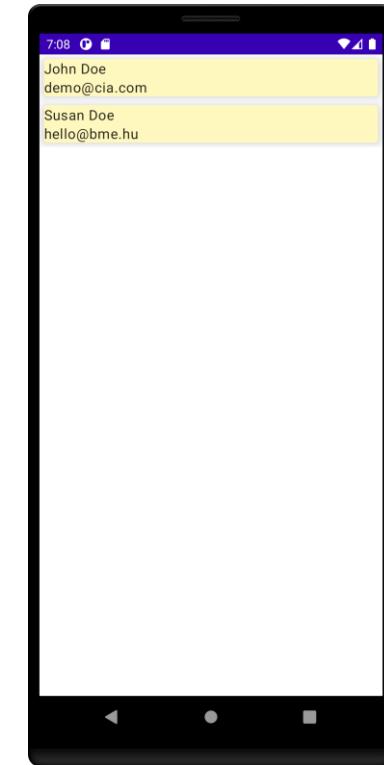
Lista
elem
nézet

```
@Composable
fun StudentCardSimple(name: String, email: String) {
    Card(
        modifier = Modifier
            .fillMaxWidth()
            .padding(5.dp),
        elevation = 10.dp,
        backgroundColor = Color(255,248,190)
    ) {
        Column {
            Text(name)
            Text(email)
        }
    }
}
```

```
class DemoComposeListActivity : ComponentActivity() {

    var students = mutableListOf<Student>(
        Student("John Doe", "demo@cia.com"),
        Student("Susan Doe", "hello@bme.hu")
    )

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MaterialTheme {
                StudentsList(students)
            }
        }
    }
}
```



További anyagok

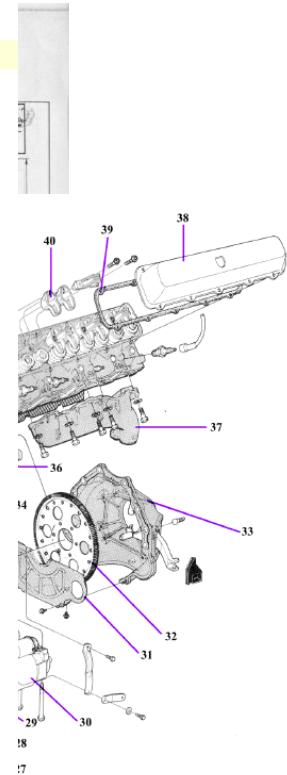
- <https://developer.android.com/jetpack/compose/tutorial>
- <https://developer.android.com/jetpack/compose/state>
- <https://foso.github.io/Jetpack-Compose-Playground/>
- <https://joebirch.co/android/exploring-jetpack-compose-card/>

Útravaló – szoftverfejlesztési elvek

Mi a mérnök feladata?



```
* upload, independent of whether the original activity is paused, stopped,  
💡 or finished.  
*/  
  
public class Activity extends ContextThemeWrapper  
    implements LayoutInflater.Factory2,  
    Window.Callback, KeyEvent.Callback,  
    OnCreateContextMenuListener, ComponentCallbacks2,  
    Window.OnWindowDismissedCallback {  
  
    private static final String TAG = "Activity";  
    private static final boolean DEBUG生命周期 = false;  
  
    /** Standard activity result: operation canceled. */  
    public static final int RESULT_CANCELED = 0;  
    /** Standard activity result: operation succeeded. */  
    public static final int RESULT_OK = -1;  
    /** Start of user-defined activity results. */  
    public static final int RESULT_FIRST_USER = 1;  
  
    static final String FRAGMENTS_TAG = "android:fragments";  
  
    private static final String WINDOW_HIERARCHY_TAG = "android:viewHierarchyState";  
    private static final String SAVED_DIALOG_IDS_KEY = "android:savedDialogIds";  
    private static final String SAVED_DIALOGS_TAG = "android:savedDialogs";  
    private static final String SAVED_DIALOG_KEY_PREFIX = "android:dialog_";  
    private static final String SAVED_DIALOG_ARGS_KEY_PREFIX = "android:dialog_args_";  
  
    private static class ManagedDialog {  
        Dialog mDialog;  
        Bundle mArgs;  
    }  
    private SparseArray<ManagedDialog> mManagedDialogs;  
  
    // set by the thread after the constructor and before onCreate(Bundle savedInstanceState) is called.  
    private Instrumentation mInstrumentation;  
    private IBinder mToken;  
    private int mIdent;  
    /*package*/ String mEmbeddedID;  
    private Application mApplication;  
    /*package*/ Intent mIntent;
```



Kihívások

- Volt már valaha dolgunk rossz kóddal?
 - „Olvasni a kódot több idő, mint írni”
- Volt már olyan eset, hogy nem volt idő egy feladat „szakszerű” megoldására, a kód tisztítására, a rövid határidő miatt?

„Ahogy nő a kód mennyisége, úgy csökken a fejlesztők produktivitása.”





Code rot (kód romlás)

- Az alkalmazások tipikusan letisztult, tiszta architektúrával indulnak
- Mi történik egy bizonyos idő után?
 - > A kód elkezd „rothadni” (romlani): kicsi hack itt-ott, egyre több *if* elágazás, mígnem az egész kódban ezek dominálnak -> átláthatatlan viselkedés
- Nehéz karbantartani, nehéz új funkciókat hozzáadni -> a fejlesztők egy idő után áttervezésért könyörögnek

Kód romlás

- A forráskód bizonyos szempontból a terv (design)
- A romlott design és a rossz architektúra tipikus tünetei
 - > Merevség
 - Folyamatosan nehezebb a kód módosítás
 - A változtatás költsége magas
 - > Törékenység
 - Apró változtatás egy modulon egy másik modulban okozhat hibás viselkedést
 - Például: egy bug javítás elront egy látszólag független részt
 - > Mozdulatlanság
 - Egy rendszer mozdulatlan, ha a részeit nem lehet könnyedén modulokba kiszervezni és máshol újra hasznosítani
 - Például: a login modul újra felhasználható legyen
 - Mozdulatlanság elkerülési stratégiák: rétegek kialakítása (adatbázis és UI különválasztása)
 - > Nyúlékonyság
 - A kód struktúra nyúlékonysága
 - Új feature implementálását könnyebb megoldani hackeléssel, mint új kód írásával/új osztály bevezetésével
 - A környezet nyúlékonysága
 - Fordítás, teszt futtatás és becheckolás körülményes és sok ideig tart

Kód romlás – Mi az okozója?

- Változó követelmények
 - > Ha olyan a kódunk/architektúránk, hogy nehéz a változásokat kezelni, az a mi hibánk
 - > A kód/architektúra rugalmas kell legyen a változások követésére és meg kell akadályozza a kód romlást
- Milyen változások miatt kezd romlani a kód? *Olyan változások, amelyek új, nem tervezett dolgokat hoznak az osztály függőségek szintjén.*
- A legtöbb tünet direkt, vagy indirekt módon a modulok közti nem megfelelő függőségre vezethető vissza.
- Az objektum orientált tervezési elvek segítenek a modulok közti függőségek kezelésében.
 - > SOLID elvek

A szoftver két értéke

- **Másodlagos érték:** a szoftver viselkedése - a szoftver azt csinálja hibamentesen, amit a felhasználó elvár
- **Elsődleges érték:** Tolerálja és egyszerűen alkalmazkodik a folyamatos változásokhoz, tehát könnyű módosítani (software is soft)



A forráskód védelme

- A forráskód a megoldásunk igazi értéke
- Android esetén a visszafejtés veszélye komoly
- Védelmi lehetőség:
 - Obfuscation, titkosítás
- ProGuard-obfuscation:
 - Tömörítés
 - Felesleges kód törlése
 - Optimalizáció
 - Osztályok, attribútumok, metódusok átnevezése



Android fejlesztési javaslatok

- Hibamentes, hatékony működés
- Megfelelő osztálykönyvtárak ismerete és használata
- Fejlesztőkörnyezet kialakítása
 - > Verziókezelés
 - > Continous Integration
 - > Tesztelés
 - Unit tesztek
 - Integrációs tesztek
 - Teszt környezet
- Clean code
 - > Kódminőség
 - > Kód újrafelhasználhatóság
 - > SOLID tervezési elvek
 - > Refaktor
- Optimalizálás
 - > Memória, CPU használat és ... energiafogyasztás
- Test Driven Development



Kódolási megfontolások

- Osztályok, függvények mérete
- Elnevezés
- Csomagok elnevezése
- Kommentezés (kell egyáltalán?)
 - > Kikommentezett kódrészek?
- Egységes formázás
- Kódolási konvenciók

Mi az objektum orientáltság?

- Az OO tulajdonképpen üzenetek továbbításáról szól: reméljük, hogy a fogadó fél megfelelően kapja meg/kezeli az üzenetet
- Az OO programozás a valós világ modellezése
- OO mechanizmusok: öröklés, polimorfizmus, egységbe zárás
- Az OO fontos minőségi jellemzője: az a képesség, hogy a függőségeket úgy fordítsa meg, hogy magas szintű logika ne függjön az alacsony szintű részletekről
- Az OO tervezés gyakorlatilag a függőségek kezeléséről szól

Függőség kezelő OO elvek: SOLID elvek

- Single Responsibility Principle (S.R.P.)
- Open Closed Principle (O.C.P.)
- Liskov Substitution Principle (L.S.P.)
- Interface Segregation Principle (I.S.P.)
- Dependency Inversion Principle (D.I.P.)

Mit értünk TDD alatt?

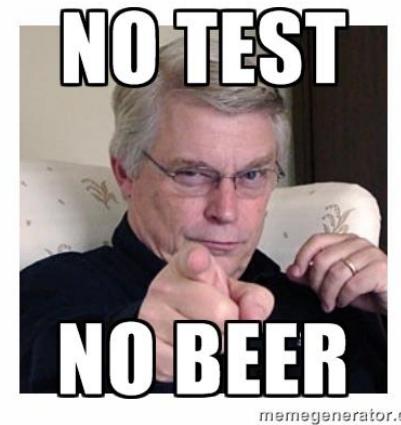
- Hagyományos fejlesztési ciklus (hosszú ciklus, több hónapos):



- Test-driven development ciklus (ismétlődő rövid ciklusok, pár perc):



- A TDD egy programozási technika, ami a következő gondolaton alapul: csak egy bukó teszt kizöldítésére írunk production kódot
- A TDD három szabálya:
 1. Írunk egy elbukó tesztet
 2. Írj csak annyi production kódot, ami kizöldíti a tesztet
 3. Tisztítsd (refactor) a tesztet és a production kódot
- Red-green-refactor



Code Kata: TDD

- Mi az a „Kata”? ☺
 - > Fogalom a karate-ban: detailed patterns of movements practiced either solo or in pairs.
- Készítsünk barkóbát TDD-vel!
- Ne feledkezzünk el a 3 szabályról:
 1. Írunk egy elbukó tesztet
 2. Írj csak annyi production kódot, ami kizöldíti a tesztet
 3. Tisztítsd (refactor) a tesztet és a production kódot

Modern fejlesztő eszközök

- Continuous Integration
 - > Jenkins
 - > Project status
- Kódminőség ellenőrzés
 - > Sonar
- Elosztott verziókezelő!
 - > Pl.: GIT



sonar



Cserkész szabály

- Nem elég jól megírni a kódot, hanem tisztán is kell tartani
- „Hagyd a tábor tisztábban, mint ahogy kaptad!”

Irodalom

- Agile Software Development by Robert C. Martin
- Clean Code Video Series by Robert C. Martin
- www.objectmentor.com



Android interjú kérdések

1. Describe the APK format.
2. What are the different phases of the Activity life cycle?
3. What is the significance of the .dex files?
4. What is the difference between Service and Thread?
5. What is a Content Provider?
6. When does ANR occur?
7. What is the difference between a regular bitmap and a nine-patch image?

<https://github.com/MindorksOpenSource/android-interview-questions>

Hogyan tovább...

- Googel I/O Extended események
- Android fejlesztői lista
 - > ~700 tag
 - > Android fejlesztői közösség
 - > Ötletek megosztása
 - > Problémák közös megoldása
 - > Értesítés különféle eseményekről
- <https://groups.google.com/group/bme-android>

Félév összefoglalása

- Android bevezetés
- Kotlin alapok
- Activity életciklus
- Felhasználói felület, layoutok, fragmentek, animációk, stb.
- Intent
- BroadcastReceiver
- Engedélyek kezelése
- Perzisztens adattárolás (SharedPreferences, File, ORM – Room)
- Hálózati kommunikáció
- Helyfüggő szolgáltatások
- Multimédia
- Services
- ContentProvider
- Grafikonok
- Szenzorok
- Játékfejlesztés
- Bluetooth, NFC, Nearby API
- Wear
- JetPack

Köszönöm a figyelmet

