UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

Mobil alkalmazásfejlesztés - Hatékony adatátvitel, feladatkezelés

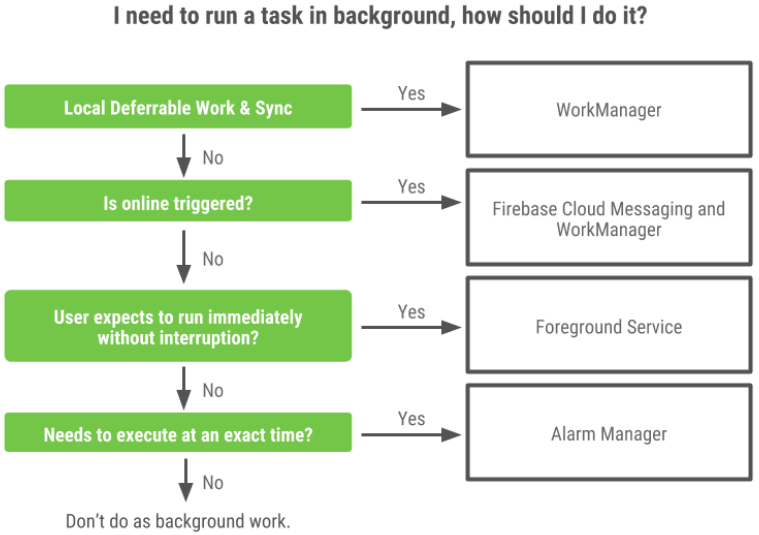
Dr. Bilicki Vilmos

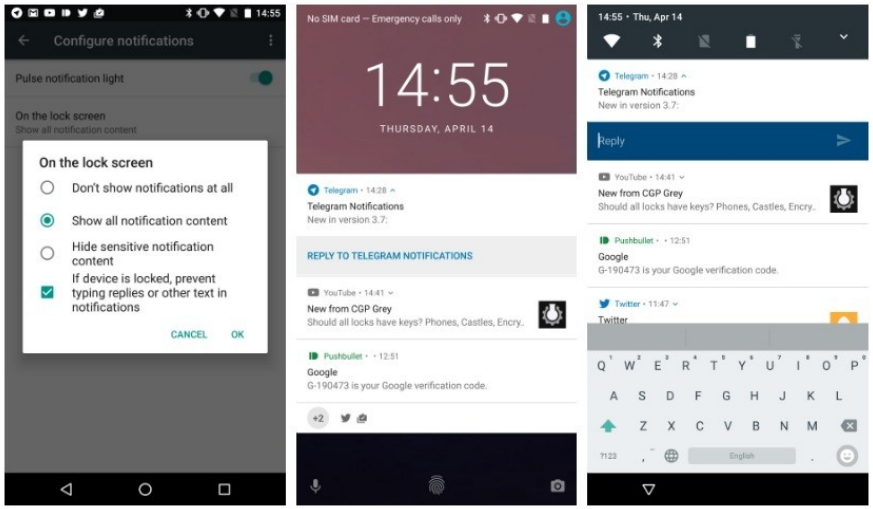
Szoftverfejlesztés Tanszék

A fóliákhoz felhasznált anyagok:
Google: Android Developer Fundamentals (Version 2), https://developer.android.com/courses/fundamentals-training/overview-v2

Az előző előadás

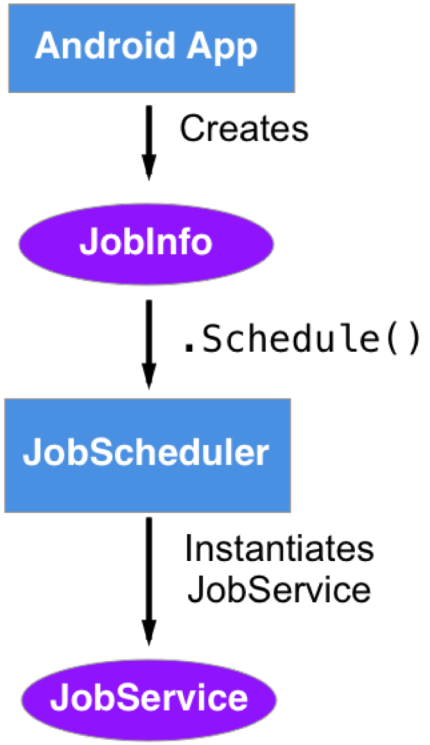
- Értesítések
- Riasztások

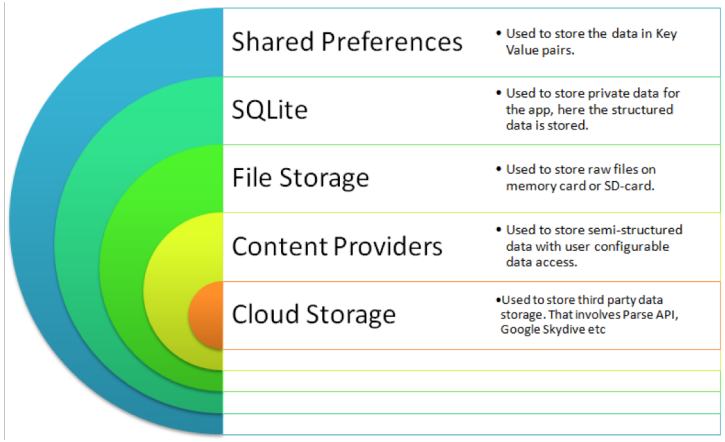




Áttekintés

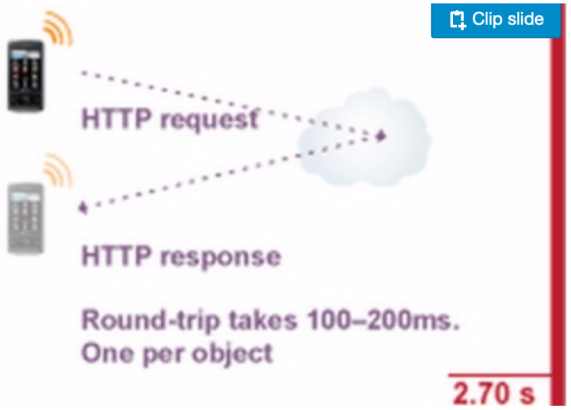
- Hatékony adatátvitel
- Job Scheduler
  - `JobService`
  - `JobInfo`
  - `JobScheduler`
- Adattárolás
  - `Android File System`
  - `Internal Storage`
  - `External Storage`
  - `SQLite Database`
  - `Other Storage Options`





Adatátvitel erőforrás igényes

- A vezetékmentes rádiós interfész sok energiát használ
  - Az eszköz lemerül.
  - Az eszközt tölteni kell.



- Az adatátvitel pénzbe kerülhet
  - A felhasználónak valós pénzbe krülhet (az ingyenes

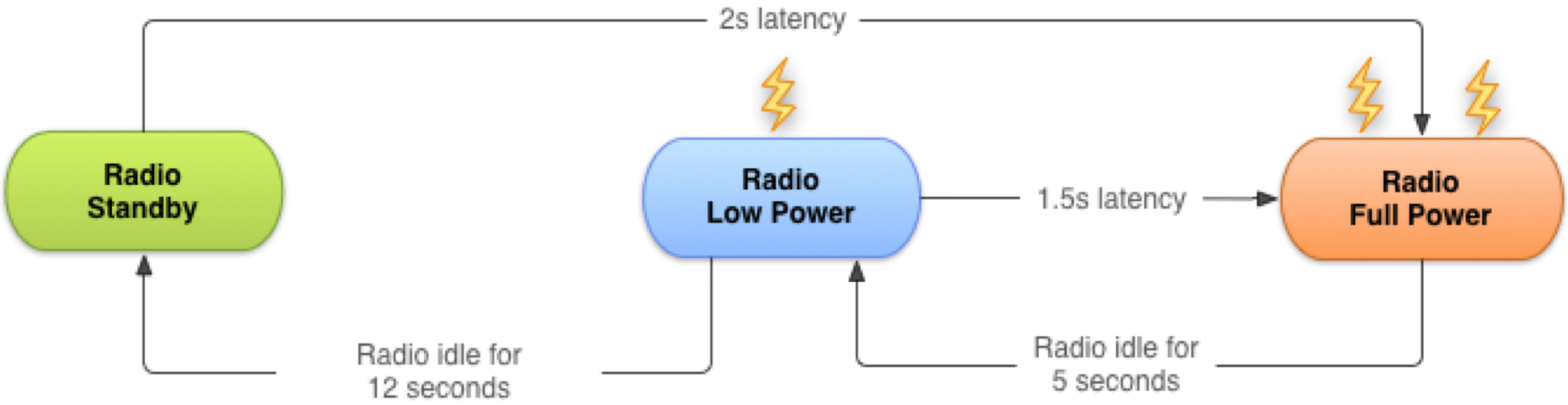
Workload	Power (% of total)							Battery life [hours]
	GSM	CPU	RAM	Graphics	LCD	Backlight	Rest	
Suspend	45	19	4	13	1	0	19	49
Casual	47	16	4	12	2	3	16	40
Regular	44	14	4	14	4	7	13	27
Business	51	11	3	11	4	11	10	21
PMD	31	19	5	17	6	6	14	29

Table 13: Daily energy use and battery life under a number of usage patterns.

Vezetékmentes interfész teljesítmény állapotai

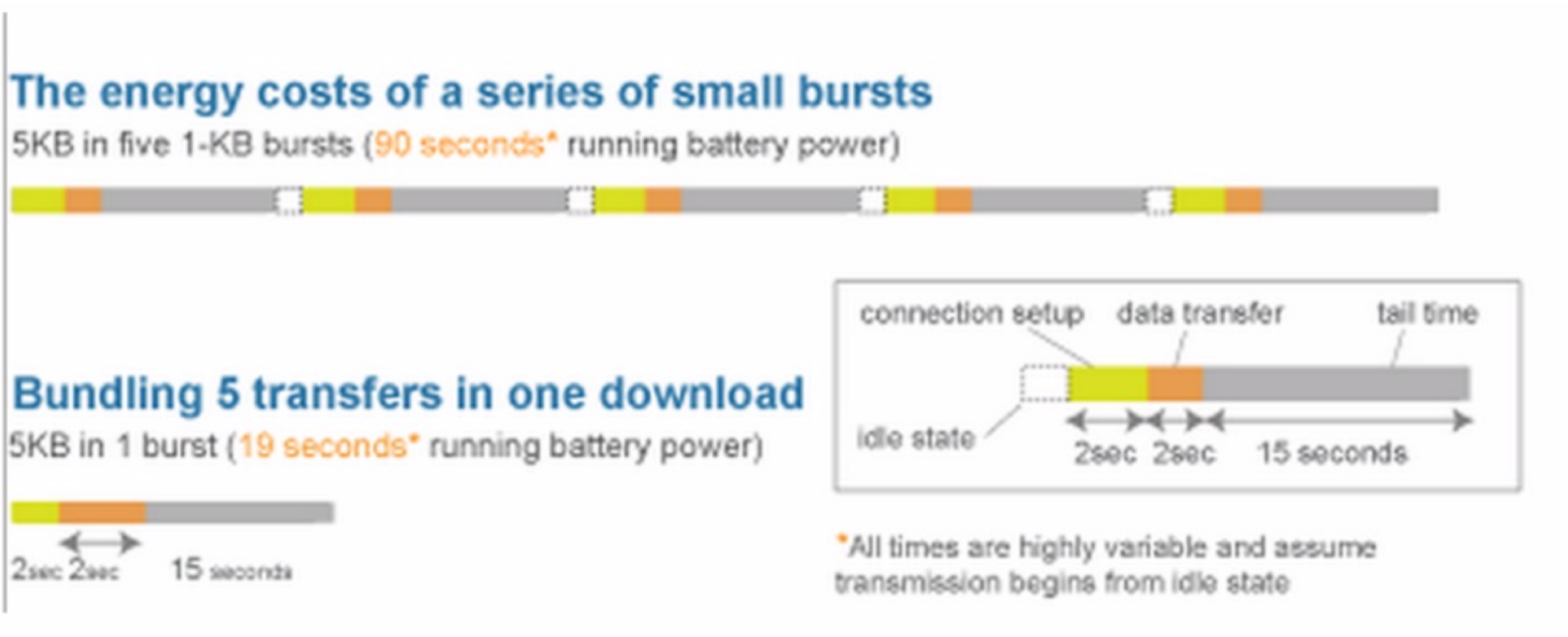
- Teljes teljesítmény—Aktív kapcsolat maximális adatátivteli sebesség..
- Alcsony teljesítmény—Köztes állapot ahol 50%-al kevesebb energiát fogyaszt.
- Várakozó—Nincs aktív hálózati kapcsolat, minimális energia.

3G interfész állapot átmenetei

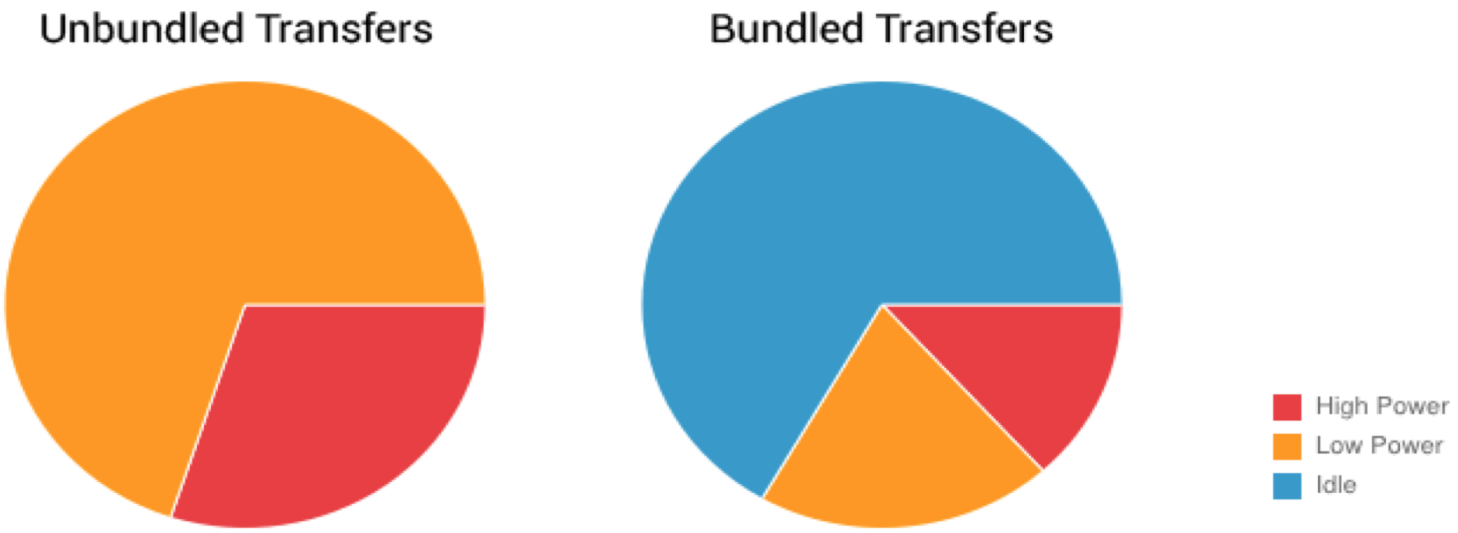


Kötegeljük az átviendő adatokat

- Egy átllagos 3G eszköznek egy-egy adatátviteli kapcsolat 20 másodpercig fogyasztja az energiát.
- Amennyiben 1 másodpercnyi adatot küldöm 18 másodpercenként akkor a rádió gyakorlatilag folyamatosan teljes teljesítményen lesz.
- Amennyiben 3s-es kötegekben küldjük akkor többnyire alvó állapotban
- Kötegeljük az átviendő információt

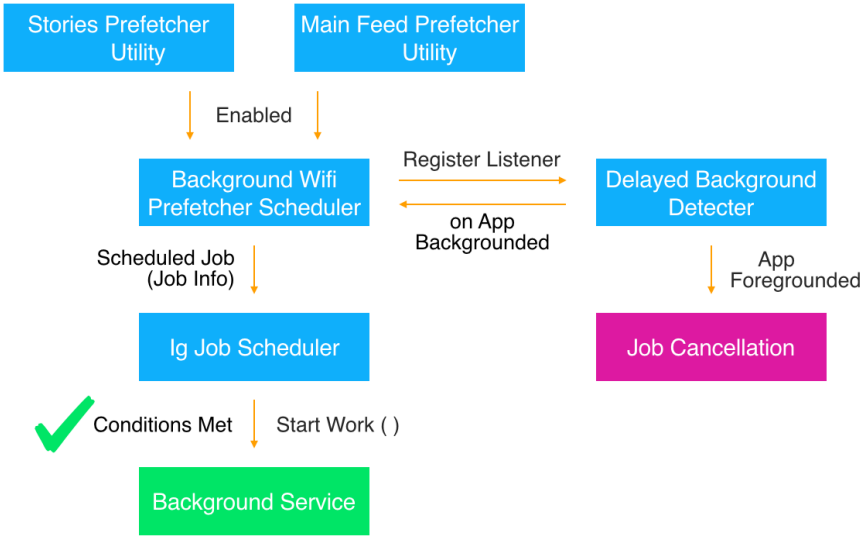


Kötelgelt vs atomi



Előre letöltött adat

- A közeljövőben szükséges adatot töltsük le előre, teljes sebességgel, egy kötegben.
- Amennyiben jól tippeltünk akkor így csökkentjük a késleltetést és az aku használatot.
- Amennyiben rosszul tippeltünk akkor feleslegesen merítettük az akumlátort és a sávszélességet.



Vegyük figyelembe a hálózati kapcsolatot

- A Wi-Fi rádiós interfész kevesebb energiát fogyaszt és nagyobb a sávszélessége mint a GSM interfésznek.
- A `ConnectivityManager` segítségével tudjuk megnézni a kapcsolat típusát és ezzel tudjuk a startégiánkat finomítani

- Phone connected via TCP port 5228
- Periodic heartbeat keeps the connection alive.

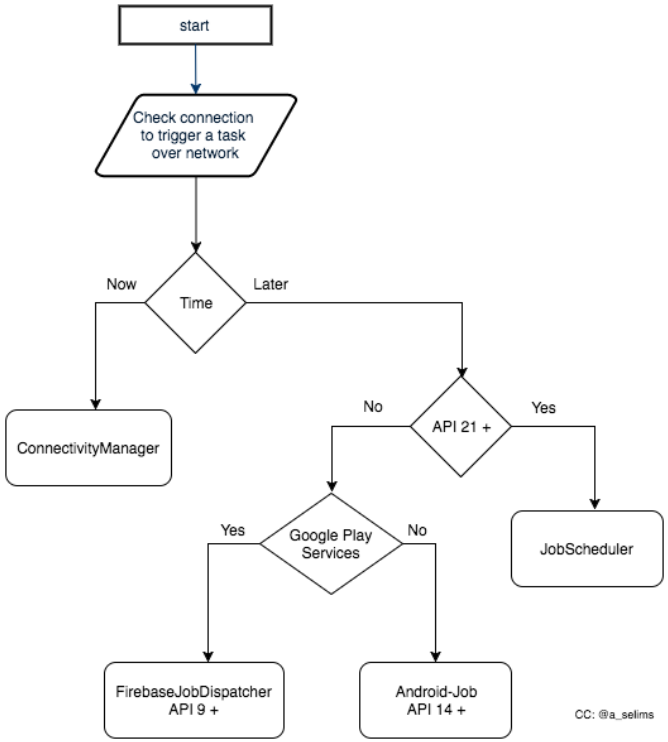
15 min on Wifi and 28 min on cell

Akkumulátor állapota

- Várjuk meg a megfelelő alkalmat az energia intenzív művetekhez.
- A `BatteryManager` üzenetszórás csatornán küldi ki a megfelelő töltöttségi, töltési állapotokat tartalmazó `Intent`-t
- Íratkozzunk fel a `BroadcastReceiver` akkumulátor csatornájára

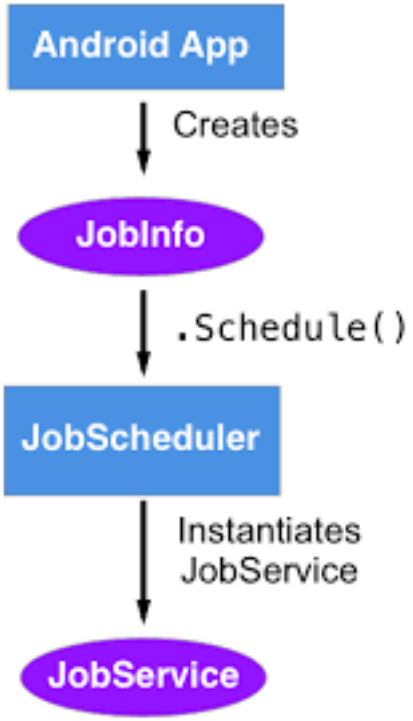
Job Scheduler – Munka Ütemező

- Segít a háttér folyamatokat intelligensen ütemezni.
- Az időzítés helyet/mellett egyébb feltételeket is figyembevesz
- Egyes esetekben `sokkal hatékonyabb` az `AlarmManager-nél`
- A feladatokat kötelegeli az akkumulátor használat minimalizálása érdekében.
- API 21+-től



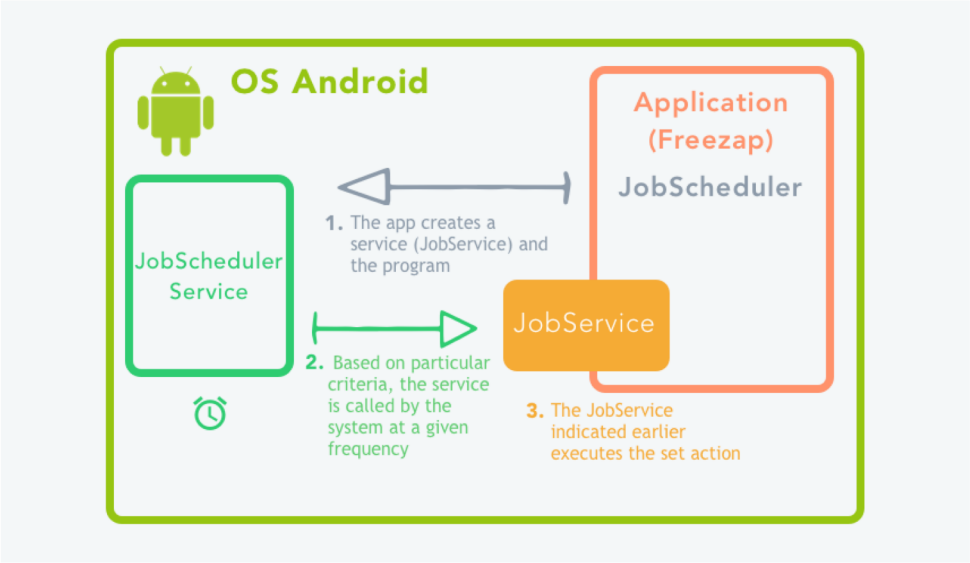
Job Scheduler komponensek

- `JobService`—A feladatot megvalósító szolgáltatás osztály.
- `JobInfo`—A feladat indítási feltételeit leíró osztály.
- `JobScheduler`—Feladatok menedzsmentje, indítása, ütemezése, leállítása



JobService

- `JobService` alosztályaként valósíthatjuk meg feladatunkat
- Írjuk felül
  - `onStartJob()`
  - `onStopJob()`
- **A fő szálban fut**



onStartJob()

- Az elvégzendő munkát tegyük ide
- A feltételek teljesülésekor hívja meg a rendszer
- A fő szálban fut
- **A komolyabb CPU időt igénylő feladatokat szervezzük ki másik szálba.**

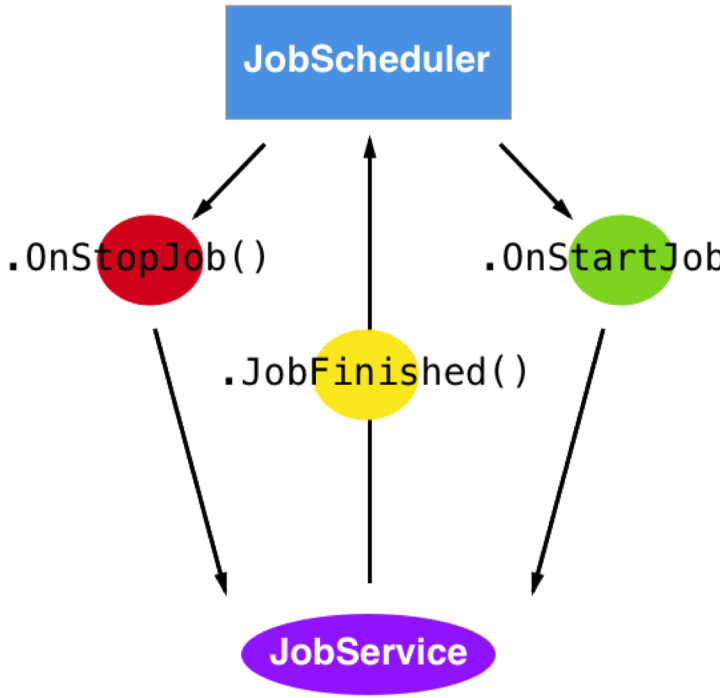
onStartJob() egy boolean-t ad vissza

Mikor fejeződik be a JobService ?

FALSE—Munka befejezve.

TRUE

- A munkát áttették másik szálba (dolgozó).
- A dolgozó szálból kell a **`jobFinished()`** metódust meghívni.
- A `JobParams` paramétereket a `onStartJob()-ban` adhatjuk át



onStopJob()

- Akkor hívják meg amikor a rendszer úgy gondolja, hogy le kell álítani a feladat végrehajtását.
- … mert adott feltételelek nem adottak.
- Nincs például Wi-Fi kapcsolat vagy az eszköz nincs nyugalmi állapotban
- A `jobFinished(JobParameters, boolean)` előtt
- Adjunk vissza `TRUE-t az újraütemezéshez`

Egyszerű JobService kód

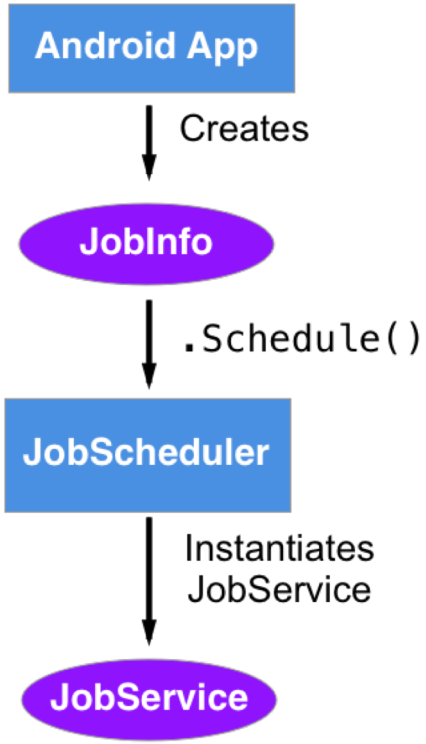
```
public class MyJobService extends JobService {
    private UpdateAppsAsyncTask updateTask = new
UpdateAppsAsyncTask();
    @Override
    public boolean onStartJob(JobParameters params) {
        updateTask.execute(params);
        return true; // work has been offloaded
    }
    @Override
    public boolean onStopJob(JobParameters jobParameters) {
       return true;
    }
}
```

Regisztráljuk a JobService-t

```
<service
android:name=".NotificationJobService”
android:permission=
"android.permission.BIND_JOB_SERVICE"/>
```

JobInfo

- A futtatáshoz szükséges feltételek leírása
- `JobInfo.Builder` objektum.



JobInfo létrehozó objkektum

- Arg 1: Job ID
- Arg 2: Szolgáltatás komponens
- Arg 3: Az indítandó `JobService`

```
JobInfo.Builder builder = new JobInfo.Builder(
        JOB_ID,
        new ComponentName(getPackageName(),
        NotificationJobService.class.getName()));
```

Feltételek


```
setRequiredNetworkType(int networkType)
setBackoffCriteria(long initialBackoffMillis, int backoffPolicy)
setMinimumLatency(long minLatencyMillis)
setOverrideDeadline(long maxExecutionDelayMillis)
setPeriodic(long intervalMillis)
setPersisted(boolean isPersisted)
setRequiresCharging(boolean requiresCharging)
setRequiresDeviceIdle(boolean requiresDeviceIdle)
```

setRequiredNetworkType()

`setRequiredNetworkType(int networkType)`

- `NETWORK_TYPE_NONE`—Alapértelmezett, nincs szükség hálózatra.
- `NETWORK_TYPE_ANY`—Kell hálózat.
- `NETWORK_TYPE_NOT_ROAMING`—Nem vándorló hálózat kel.
- `NETWORK_TYPE_UNMETERED`—Ingyenes hálózat kell

setMinimumLatency()

`setMinimumLatency(long minLatencyMillis)`

- A befejezés előtt megvárt minimális milliszekundumok száma.

setOverrideDeadline()

`setOverrideDeadline(long maxExecutionDelayMillis)`

- Mennyiben nem teljesülnek a feltételek akkor max ennyi miliszekundumig vár.

setPeriodic()

`setPeriodic(long intervalMillis)`

- Adott idő után megismételi afeladatot.
- Az ismétlési időtartamot kel megadni.
- Kizárja a min és max várakozási időket.
- Nem garantált, hogy lefut.

setPersisted()

`setPersisted(boolean isPersisted)`

- Az újraindulásnál legyen perzisztens, vagy nem.
- Igaz/Hamis
- Szükséges a `RECEIVE_BOOT_COMPLETED` engedély

setRequiresCharging()

`setRequiresCharging(boolean requiresCharging)`

- Az induláshoz/futáshoz szükséges a töltő kapcsolat.
- Igaz/Hamis
- Alapéretelmezett: Hamis

setRequiresDeviceIdle()

`setRequiresDeviceIdle(boolean requiresDeviceIdle)`

- Az eszköznek tétlen állapotban kell-e lennie
- A tétlen egy laza definíció, nagyjából azt jelneti, hogy egy ideje nem használják.
- Erőforrás igényes feladatoknál használjuk.
- Igaz/Hamis, alapértelmezett: Hamis

JobInfo kód

```
JobInfo.Builder builder = new JobInfo.Builder(
        JOB_ID, new ComponentName(getPackageName(),
        NotificationJobService.class.getName()))
        .setRequiredNetworkType(JobInfo.NETWORK_TYPE_UNMETERED)
        .setRequiresDeviceIdle(true)
        .setRequiresCharging(true);
JobInfo myJobInfo = builder.build();
```

A munka ütemezése

1. Kérjük el a `JobScheduler` objektumot a rendszertől
2. Hívjuk meg a `schedule()` metódust egy `JobScheduler` segítségével egy `JobInfo` objektummal.

```
mScheduler =
    (JobScheduler)getSystemService(JOB_SCHEDULER_SERVICE);
mScheduler.schedule(myJobInfo);
```

Adattárolás

- Android fájlrendszer
- Belső tároló
- Külső tároló
- SQLite adatbázis
- Egyébb tárolási megoldások

Adattárolás

- Shared Preferences—Egyszerű, privát név-érték pár tároló
- Internal Storage—Az eszköz memórijában használható privát tátroló
- External Storage—Az eszközön vagy külső táron lévő publikus tár
- SQLite Databases—Adatbázisba mentett struktúrált infomráció
- Content Providers—Pritvát tár, publikálási lehetőséggel

Adattárolás az Androidon túl

- Network Connection—A weben saját szerveren
- Cloud Backup—A felhőben, mentésként
- Firebase Realtime Database—Tároljuk és szinkronizáljuk a kliensek között az adatot valós időben egy felhő alapú NoSQL adatbázisban

Android fájlrendszer

- Külső tár – Publikus könyvtárak
- Belső tár– Pritvát csak azz app számára látható fájlok

Az alkalmazások böngészhetik a fájlokat, mappákat

A struktúr és a műveletek hasonlítanak a lLinux és java.io megközelítésre

Belső tároló

- Mindég rendelkezésre áll
- Az eszköz fájlrendszerét használja
- Csak az adott alkalmazás írhatja, olvashatja (beállítható, hogy más is)
- Amikor az alkalmazást törlik akkor az összes kapcslódó fájl is törtlődik

Külső tároló

- Nem érhető el mindég, kivehető
- Az eszköz vagy egy kiegészítő kártya fájlrendszerét használja
- Bármelyik alkalmazás hozzáférhet
- Az alkalmazás törlésekor megmarad

Külső vagy belső tárolót használjuk?

A belső a célszerű amikor:

- sem a felhasználó, sem más alkalmazásnak nem szeretnénk közvetlen hozzáférést biztosítani

A külső a célszerű amikor

- amikor nem akarjuk a láthatóságot korlátozni
- meg szertenénk osztani más alkalmazásokkal
- hozzáférést szeretnénk adni a felhasználónak és a számítógépének

A felhasználói fájlokat mentsük a publikus tárolóba

- A külső vagy a felhasználó által létreozott fájlokat célszerű a publikus tárolóba kezelni
- Így
  - Más programok is hozzáférnek
  - A felhasználó elérheti PC-ről

Belső tár

- Privát, csak az alkalmazásnak látható könyvtárak
- Az alkalmazásnak mindég van írás/olvasás joga
- Permanens könyvtár—getFilesDir()
- Ideiglenes könyvtár—getCacheDir()

Fájl létrehozása

```
File file = new File(
        context.getFilesDir(), filename);
```

Használjuk a szabványos java.io fájl műveleteket

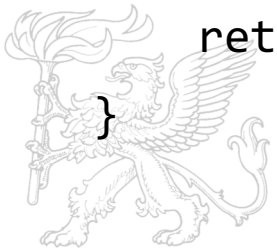
Külső tár

- Az eszközön vagy SD kártyán
- A jogosultságokat az Android Manifest-ben kérhetjük
  - Az írás engedély magába foglalja az olvasást is

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

Rendelkezésre álló tárhely


```
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}
```



Példa külső könyvtárak

- DIRECTORY_ALARMS és DIRECTORY_RINGTONES
- DIRECTORY_DOCUMENTS
- DIRECTORY_DOWNLOADS

developer.android.com/reference/android/os/Environment.html

Publikus könyvtárak elérése

1. Kérjük el az útvonalat [getExternalStoragePublicDirectory()]
2. Hozzuk létre a fájlt

```
File path = Environment.getExternalStoragePublicDirectory(
        Environment.DIRECTORY_PICTURES);
File file = new File(path, "DemoPicture.jpg");
```

Mennyi helyünk maradt?

- Amennyiben nincs elég hely akkor IOException-t kapunk
- Amennyiben tudjuk az igényünket akkor validáljuk
  - getFreeSpace()
  - getTotalSpace().
- Ha nem akkor kezeljük le
  - try/catch IOException

Fájlok törlése

- Külső

```
myFile.delete();
```

- Belső

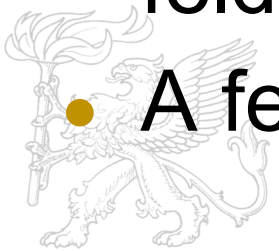
```
myContext.deleteFile(fileName);
```

Ne töröljük a felhasználó fájlait!

Amikor a flehasználó törli az alkalmazást akkor a belső tár is törlődik. **Ne tároljunk ott a felhasználónak fontos adatokat!**

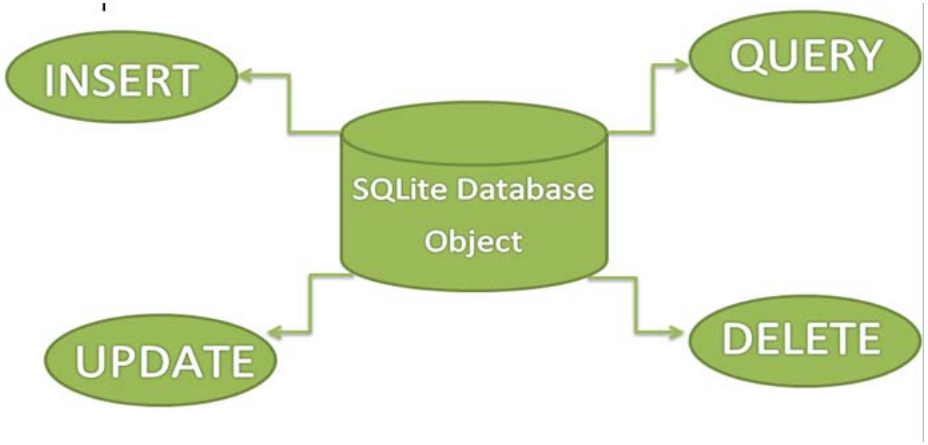
Példa

- Adott alkalmazással létrehozott és feldolgozott fényképek
- A felhasználó által megvásárolt zenék



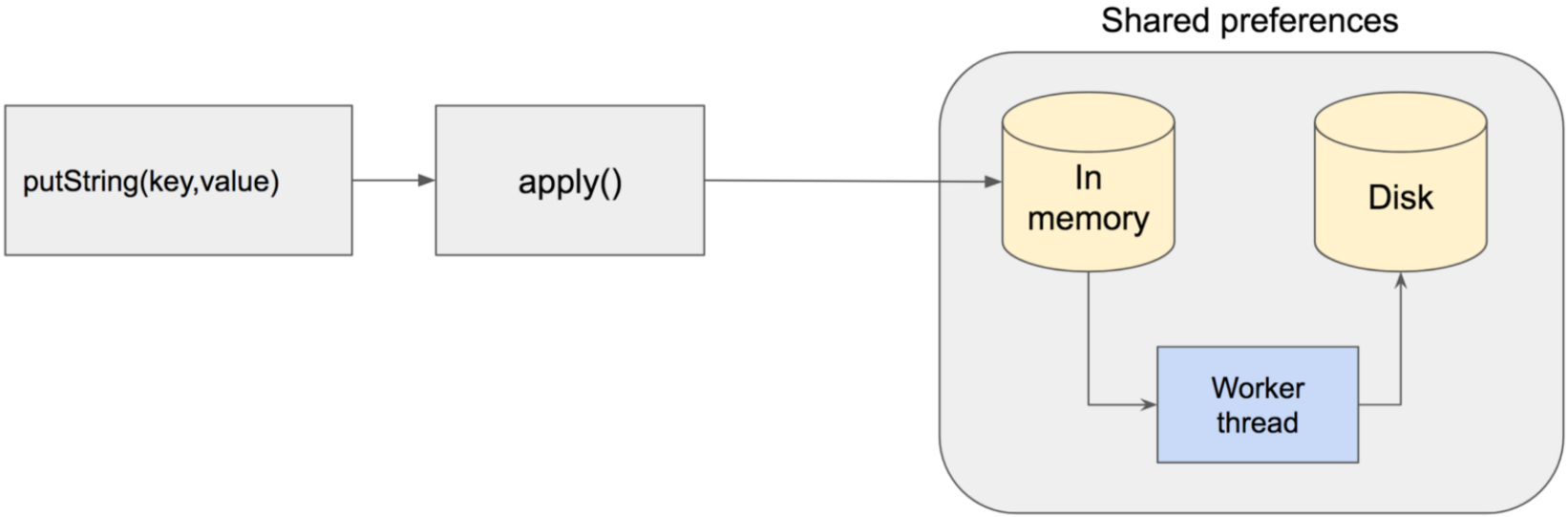
SQLite adatbázis

- Struktúrált, ismétlődő adatok tárolására
- Az Android egy SQL szerű adatbázist biztosít erre



Shared Preferences

- Kis mennyiségű egyszerű adat írása/olvasása az eszköz tárhelyén



Firebase alkalmazása

Adatok tárolása, szinkronizálása a Firebase felhő adatbázissal

A kliensek között szinkronizálható, offline képse

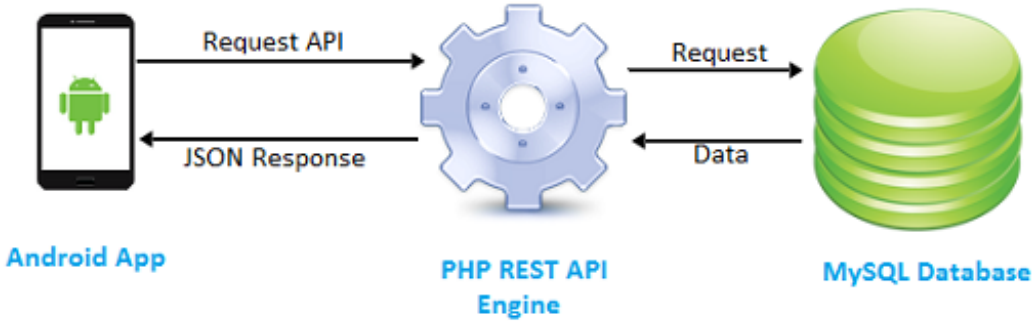
firebase.google.com/docs/database/

Firebase Valósidejű Adatbázis

- A kapcsolt alkalmazások adatcsere helyszíne
- Felhőben futtatva
- Az adat JSON-ban tárolva
- Valósidajeű szinkronizálás a kliensek között

Hálózati kapcsolat

- Web alapú saját háttértár alkalmazása
- Az alábbi csomagokra építhetünk
  - java.net.*
  - android.net.*

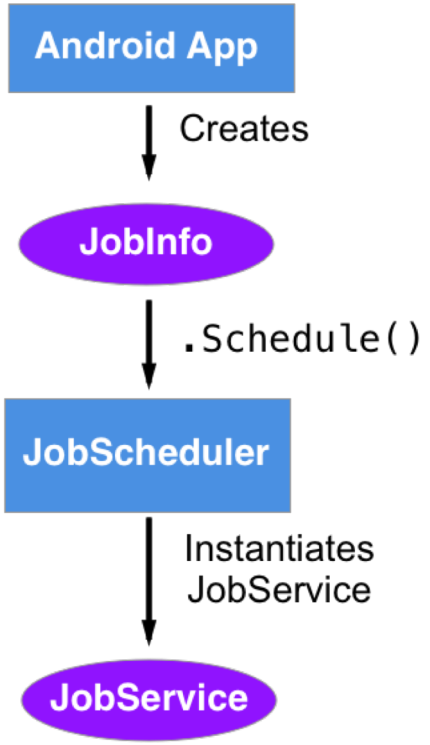


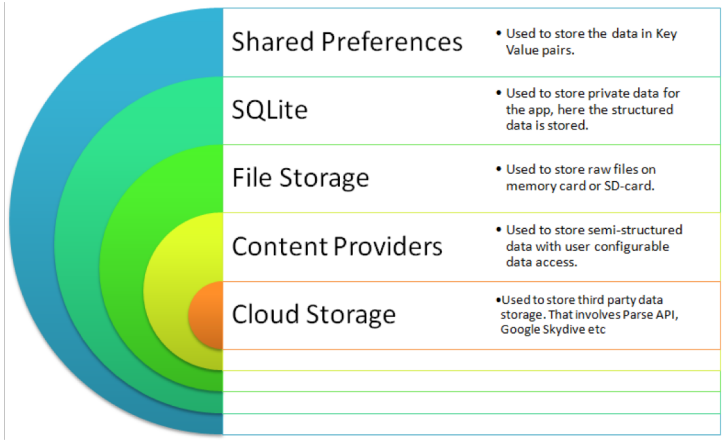
Adatok mentése

- Auto Backup a 6.0 (API level 23) -tól
- Az adatot automatikusan menti a flehőbe
- Nem kell programozni és ingyenes
- Tesreszabható alkalmazásonként
- Továbbiak Configuring Auto Backup for Apps

Áttekintés

- Hatékony adatátvitel
- Job Scheduler
  - `JobService`
  - `JobInfo`
  - `JobScheduler`
- Adattárolás
  - `Android File System`
  - `Internal Storage`
  - `External Storage`
  - `SQLite Database`
  - `Other Storage Options`





A következő előadás tartalma

1. Shared perefernces
2. Alkalmazás beállítások
3. SQL Lite
4. SQLLite bevezető

56