Senzory

Táto prednáška sa zaoberá témou vstavaných senzorov v systéme Android ale aj lokalizačným podsystémom Location API.

Úvod do senzorov

Väčšina dnešných Android zariadení obsahuje vstavané senzory, ktoré umožňujú merať zrýchlenie, natočenie zariadenia a mnoho ďalších fyzikálnych veličín. Tieto senzory sú schopné poskytnúť surové dáta s veľkou presnosťou a vo veľkom rozsahu a sú užitočné, ak chceme monitorovať pohyb alebo polohu zariadenia v trojrozmernom priestore alebo ak chceme monitorovať zmeny okolitého prostredia v blízkosti zariadenia. Napríklad hra môže sledovať údaje z akcelerometra a identifikovať zložité gestá a pohyby používateľa, ako napríklad naklonenie, chvenie, rotácia a pod. Podobne aj aplikácia na počasie môže na výpočet a určenie rosného bodu použiť snímač teploty a vlhkosti alebo navigácia môže využiť dáta z trojosového magnetometra, ktorý sníma geomagnetické pole Zeme a dáta z akcelerometra na spresnenie navigácie.

Platforma Android podporuje tri kategórie senzorov:

- Pohybové senzory

Tieto senzory merajú fyzikálne veličiny ako: zrýchlenie a rotačné sily pozdĺž troch osí. Do tejto kategórie patria akcelerometre, gravitačné senzory, gyroskopy a senzory natočenia.

- Senzory prostredia

Tieto senzory merajú rôzne parametre prostredia, ako je teplota a tlak okolitého vzduchu, osvetlenie a vlhkosť. Táto kategória zahŕňa barometre, fotometre (meranie intenzity svetla) a teplomery.

- Polohové senzory

Tieto senzory merajú fyzickú polohu zariadenia. Táto kategória zahŕňa senzory orientácie a magnetometre.

V systéme Android môžeme senzory ešte rozdeliť podľa toho, či údaje pochádzajú priamo z fyzického senzora alebo ide o softvérový senzor, ktorý poskytuje už predspracované dáta. Predspracovanie dát môže zahŕňať napríklad: filtrovanie dát, fúziu dát z rôznych senzorov alebo transformovanie dát do inej formy. Hardvérové senzory odvodzujú údaje priamym meraním špecifických vlastností prostredia.

Medzi hardvérové senzory radíme:

- Mikrofón
- Kamera
- Teplota
- Poloha (GPS)
- Accelerometer
- Proximity
- Tlak
- Svetelnosť

Medzi softvérové radíme:

- Gravitácia

- Lineárna akcelerácia
- Orientácia

Android poskytuje rozsiahly framework na prístup k senzorom. Tento framework poskytuje niekoľko tried a rozhraní, ktoré môžeme využiť napríklad na:

- Určiť aké senzory sú k dispozícii na danom zariadení. Každý výrobca zariadení si sám určuje akým senzormi osadí svoje zariadenia. Nie je to štandardizované.
- Určiť vlastnosti jednotlivých snímačov, napríklad maximálny dosah, výrobca, požiadavky na napájanie a rozlíšenie.
- Získať surové údaje zo snímača a definujte **minimálnu** vzorkovaciu frekvenciu (ako často dostávame dáta). Slovíčko minimálne je použité z toho dôvodu, že senzory sú zdielané medzi všetkými aplikáciami zariadenia. V prípade ak niektorá aplikácia požaduje vyššiu vzorkovaciu frekvenciu ako postačovala pre našu aplikáciu, tak vzorkovacia frekvencia sa zvýši, čo ovplyvni aj vzorkovanie pre našu aplikáciu. Hodnota vzorkovacej frekvencie ale nemôže klesnúť pod nami nastavené minimum.
- Zaregistrovanie a zrušenie registrácie na odoberanie udalostí od senzora. Príklady udalostí, na ktoré sa môžeme registrovať sú: nová hodnota, zmena presnosti, zmena citlivosti a pod.

Málo zariadení so systémom Android má každý typ senzora. Napríklad väčšina mobilných telefónov a tabletov má akcelerometer a magnetometer, ale menej prístrojov má barometre alebo teplomery. Zariadenie môže mať tiež viac ako jeden senzor daného typu. Napríklad zariadenie môže mať dva snímače gravitácie, z ktorých každý má iný rozsah.

senzor	typ	popis	Bežné použitia
`TYPE_ACCELEROMETER`	hardvér	Zmeria silu zrýchlenia v m/s^2 , ktorá pôsobí na zariadenie vo všetkých troch osách (x, y a z), vrátane gravitačnej sily.	Detekcia pohybu (chvenie, náklon atď.).
`TYPE_AMBIENT_TEMPERATURE`	hardvér	Zmeria teplotu okolia v stupňoch Celzia (°C). Pozri poznámku nižšie.	Monitorovanie teploty vzduchu.
`TYPE_GRAVITY`	Softvér alebo hardvér	Zmeria gravitačnú silu v m/s^2, ktorá pôsobí na zariadenie na všetkých troch fyzických osách (x, y, z).	Detekcia pohybu (chvenie, náklon atď.).
`TYPE_GYROSCOPE`	hardvér	Zmeria rýchlosť rotácie zariadenia v rad / s okolo každej z troch fyzických osí (x, y a z).	Detekcia rotácie (rotácia, otočenie atď.).
`TYPE_LIGHT`	hardvér	Meria úroveň okolitého svetla (osvetlenie) v *lx*.	Ovládanie jasu obrazovky.
`TYPE_LINEAR_ACCELERATION`	Softvér alebo hardvér	Zmeria silu zrýchlenia v m/s^2, ktorá pôsobí na zariadenie vo všetkých troch osách (x, y a z), s výnimkou gravitačnej sily.	Monitorovanie zrýchlenia pozdĺž jednej osi.
`TYPE_MAGNETIC_FIELD`	hardvér	Zmeria sa okolité geomagnetické pole pre všetky tri fyzikálne osi (x, y, z) v µT.	Vytvorenie kompasu.
`TYPE_ORIENTATION`	softvér	Zmeria stupne natočenia, ktoré zariadenie vykoná okolo všetkých troch fyzických osí (x, y, z). Od API úrovne 3 môžete získať maticu náklonu a rotačnú maticu pre zariadenie pomocou gravitačného senzora a senzora geomagnetického poľa v spojení s touto metódou `getRotationMatrix()` .	Určenie polohy zariadenia.
`TYPE_PRESSURE`	hardvér	Zmeria tlak okolitého vzduchu v hPa alebo mbar.	Monitorovanie zmien tlaku vzduchu.
`TYPE_PROXIMITY`	hardvér	Zmeria blízkosť objektu v cm vzhľadom na obrazovku zariadenia. Tento senzor sa zvyčajne používa na určenie, či je slúchadlo držané pri uchu osoby.	Poloha telefónu počas hovoru.
`TYPE_RELATIVE_HUMIDITY`	hardvér	Zmeria relatívnu vlhkosť okolia v percentách (%).	Monitorovanie rosného bodu, absolútnej a relatívnej vlhkosti.
`TYPE_ROTATION_VECTOR`	Softvér alebo hardvér	Zmeria orientáciu zariadenia poskytnutím troch prvkov rotačného vektora zariadenia.	Detekcia pohybu a detekcia rotácie.
`TYPE_TEMPERATURE`	hardvér	Zmeria teplotu zariadenia v stupňoch Celzia (°C). Táto implementácia senzora sa medzi zariadeniami líši a tento senzor bol nahradený `TYPE_AMBIENT_TEMPERATURE` senzorom v API Level 14	Monitorovanie teploty.

Framework

K senzorom môžete pristupovať a získavať nespracované t.j. surové dáta pomocou frameworku ktorý je súčasťou `android.hardware` balíka a obsahuje nasledujúce triedy a rozhrania:

- `SensorManager`

Túto triedu môžete použiť na vytvorenie inštancie služby spätej so senzormi. Táto trieda poskytuje rôzne metódy pre prístup a zoznam senzorov, registráciu a zrušenie registrácie prijímania udalostí senzorov. Táto trieda tiež poskytuje niekoľko konštánt, ktoré sa používajú na definovanie presnosti senzora, nastavenie rýchlosti získavania údajov (vzorkovacia frekvencia) a kalibráciu senzorov.

- `Sensor`

Túto triedu môžete použiť na vytvorenie inštancie konkrétneho senzora. Táto trieda poskytuje rôzne metódy, ktoré vám umožňujú určiť schopnosti senzora.

- `SensorEvent`

Systém používa túto triedu na vytvorenie objektu udalosti senzora, ktorý poskytuje informácie o udalosti ktorú emitoval senzor. Objekt `SenzorEvent` obsahuje nasledujúce informácie: prvotné údaje senzora, typ senzora, ktorý generoval udalosť, presnosť údajov a časové razítko udalosti.

- `SensorEventListener`

Toto rozhranie môžete použiť na vytvorenie dvoch metód spätného volania, ktoré prijímajú oznámenia (udalosti senzora) pri zmene hodnôt senzora alebo pri zmene presnosti senzora.

Identifikovanie senzorov a ich vlastnosti

Získanie inštancie triedy `SensorManager` Tento krok predstavuje vstupný bod, ktorý je nutné spraviť, aby sme vedeli pristupovať k senzor framework-u.

```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

Pomocou inštancie triedy `SensorManager` si vieme vyžiadať zoznam dostupných senzorov. Vyžiadame si buď celý zoznam alebo iba určitú kategóriu:

- `TYPE_ALL` - všetky dostupné senzory
- TYPE_GYROSCOPE , TYPE_LINEAR_ACCELERATION , TYPE_GRAVITY`

```
List<Sensor> deviceSensors = mSensorManager.getSensorList(Sensor.TYPE_ALL);
```

Overenie či daný senzor existuje na zariadení pomocou `getDefaultSensor()` . Ak daný senzor v zariadení existuje, tak vráti inštanciu objektu `Senzor` . Ak zariadenie obsahuje viac ako jeden senzor daného typu, jeden zo senzorov musí byť označený ako predvolený senzor. Ak pre daný typ senzora neexistuje predvolený senzor, volanie metódy vráti hodnotu `null` , čo znamená, že takýto typ senzora nie je v zariadení osadený.

```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
if (mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null){
  // Success! There's a magnetometer.
  }
else {
  // Failure! No magnetometer.
  }
```

Pomocou získaného objektu `Senzor` vieme určit aj ďalšie vlastnosti konkrétneho senzora tak, že zavoláme príslušné metódy napr:

	Verejné metódy
int	getMaxDelay()
float	getMaximumRange()
int	getMinDelay()
String	getName()
float	getPower()
int	getReportingMode()
float	getResolution()
String	getStringType()
int	getType()
String	getVendor()
int	getVersion()

Vzorkovacia frekvencia

Metóda `getMinDelay()` je užitočná ak chceme získať informáciu aká je maximálna vzorkovacia frekvencia. Táto premenná vracia minimálnu periódu vzorkovania v mikrosekundách na výpočet vzorkovacej frekvencie použije známy vzorec:

$$f_{vzorkovacia} = \frac{1}{getMinDelay() \times 10^6 [Hz]}$$

Ak metóda `getMinDelay()` vráti 0 tak v tomto prípade senzor vyvolá udalosť iba pri zmene meranej veličiny

Komunikácia so senzormi

Komunikácia so senzorom prebieha asynchrónnym spôsobom. Senzory Androidu sú riadené externými službami a komunikujú na základe odosielania udalosti. Aplikácia sa musí zaregistrovať na príjem týchto udalostí implementovaním rozhrania `SensorEventListener`. V rozhraní `SensorEventListener` existujú dve metódy spätného volania tzv callback metódy `onAccuracyChanged()` a `onSensorChanged()`.

Android volá tieto metódy vždy, keď nastane táto situácia:

- Ak sa zmení presnosť senzora systém volá metódu `onAccuracyChanged()` a poskytne referenciu na `Sensor` objekt,

ktorého sa daná situácia týka.

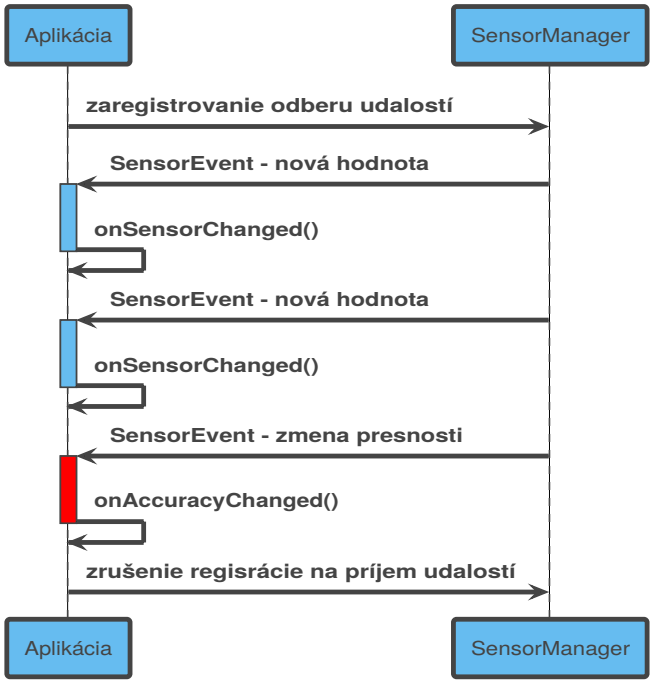
- Ak senzor načíta novú hodnotu systém volá metódu `onSensorChanged()` a poskytne referenciu na `SensorEvent`. Tento objekt obsahuje nasledujúce atribúty:

Atribúty

`public int`	`accuracy` Presnosť tejto udalosti.
`public Sensor`	`sensor` Senzor, ktorý generoval túto udalosť.
`public long`	`timestamp` Čas v nanosekundách, v ktorom sa udalosť stala
`public final float[]`	`values` Dĺžka a obsah poľa `values` závisí od toho, pre ktorý typ senzoru bol daný objekt vytvorený.

Atribút `SensorEvent.values`

Dôležité je si uvedomiť, že objekt `SensorEvent` sa používa pre všetky typy senzorov, ktoré poskytujú rôzne dáta a preto na pole hodnôt`values` sa treba pozrieť vždy v kontexte konkrétneho senzora, ktorý určuje čo uloží do daného poľa.



Zaregistrovania odberu udalostí od senzora

```
@Override
 protected void onResume() {
   super.onResume();
   mSensorManager.registerListener(this, mLight,SensorManager.SENSOR_DELAY_NORMAL);
 }
```

Pri registrácii sme definovali pomocou konštanty `SENSOR_DELAY_NORMAL`, minimálnu vzorkovaciu frekvenciu alebo inak povedané maximálny čas medzi vzorkami. Pre každý typ aplikácie je vhodne vybrať túto hodnota tak, aby zodpovedala požiadavkám, ktoré sú kladené na aplikáciu. Zároveň však treba brať do úvahy výpočtové a s tým späté aj energetické nároky, ktoré vyplývajú z použitia senzora a nastavenia veľmi vysokej vzorkovacej frekvencie. Exitujú tieto konštanty, ktoré majú napomôcť vybrať vhodnú vzorkovaciu frekvenciu: už spomenutá `SENSOR_DELAY_NORMAL` a potom

`SENSOR_DELAY_GAME` , `SENSOR_DELAY_UI` , alebo špecifická hodnota zadaná v mikrosekundách.

Ako už bolo spomenuté v predchádzajúcich častiach, vzorkovacia frekvencia sa môže meniť vzhľadom na to, že senzory sú zdielané medzi všetkými aplikáciami. Ak iná aplikácia potrebuje vyššiu vzorkovaciu frekvenciu, tak aj naša aplikácia bude dostávať údaju častejšie. Neexistuje metóda pomocou ktorej by sa dala určiť aktuálna vzorkovacia frekvencia, jediná možnosť je určiť ju nepriamo pomocou časovej známky, ktorá je obsiahnutá v objekte `SensorEvent` .

Zrušenie registrácie odberu udalostí od senzora

```
@Override
 protected void onPause() {
   super.onPause();
   mSensorManager.unregisterListener(this);
 }
```

Životný cyklus

Pri práci so senzormi nesmieme zabudnúť na životný cyklus aplikácie a preto registrovanie a odregistrovanie vykonávame v metódach životného cyklu `onResume` , `onPause` tak aby sme zbytočne nepoužívali senzory, ktoré už nepotrebujeme (znižovanie spotreby).**Systém senzory nevypína automaticky!**

Komplexný príklad použitia Frameworku na prácu so senzormi

```
public class SensorActivity extends Activity implements SensorEventListener {
    private SensorManager sensorManager;
    private Sensor mLight;

    @Override
    public final void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        mLight = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
    }

    @Override
    public final void onAccuracyChanged(Sensor sensor, int accuracy) {
        // Do something here if sensor accuracy changes.
    }

    @Override
    public final void onSensorChanged(SensorEvent event) {
        // The light sensor returns a single value.
        // Many sensors return 3 values, one for each axis.
        float lux = event.values[0];
        // Do something with this sensor value.
    }

    @Override
    protected void onResume() {
        super.onResume();
        sensorManager.registerListener(this, mLight, SensorManager.SENSOR_DELAY_NORMAL);
    }

    @Override
    protected void onPause() {
        super.onPause();
        sensorManager.unregisterListener(this);
    }
}
```

Vysporiadanie sa s rôznorodosťou Android zariadení

Na zistenie prítomnosti daného senzora v zariadení máme dve komplementárne možnosti:

- **Detegovať senzory za behu** a podľa potreby zapnúť alebo vypnúť funkcie aplikácie.

```
private SensorManager sensorManager;
...
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
if (sensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE) != null){
    // Success! There's a pressure sensor.
} else {
    // Failure! No pressure sensor.
}
```

- **Použitie filtrov v službe Google Play** - zakázanie inštalácie aplikácie už priamo v obchode Google Play.
  - Tento filter sa nastavuje v `AndroidManifest.xml` súbore pomocou xml elementu `<user-feature>`

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

  - `android:required=` určuje, či daný senzor je nevyhnutný pre fungovanie aplikácie. Hodnota `false` , dovolí

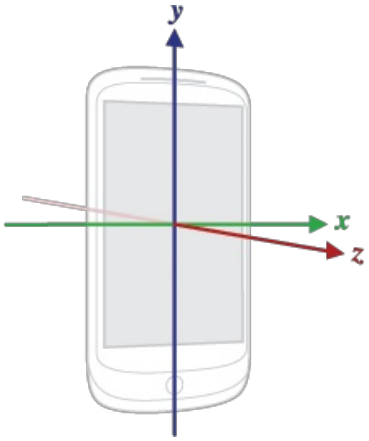
aplikáciu nainštalovať aj keď daný senzor nie je v zariadení prítomný, aplikácia sa musí tomu prispôsobiť.

> **Note**
>
> Pre prehladnosť sa odporúča všetky používané senzory aj tie nie kritické pre chod aplikácie, zapísať do `AndroidManifest.xml` súboru.

Súradnicový systém senzorov

Pri práci so senzormi, ktoré z princípu činnosti merajú dáta v osiach, Android využíva štandardný 3-osový súradnicový systém. Pre väčšinu senzorov je súradnicový systém definovaný vzhľadom na obrazovku zariadenia, keď je zariadenie držané v takzvanej natívnej polohe (pozri obrázok) . Ak je zariadenie držané v tejto polohe os X je vodorovná a ukazuje doprava, os Y je zvislá a ukazuje nahor a os Z ukazuje smerom k vonkajšej strane obrazovky. V tomto systéme majú súradnice za obrazovkou záporné hodnoty Z. Tento súradnicový systém používajú nasledujúce senzory: accelerometer, gravitačný senzor, gyroskop, lineárna akcelerácia, senzor geomagnetického poľa (kompas)

Súradnicový systém používaný v Android OS



Dôležité vlastnosti:

- Je dôležité si pamätať, že osy pri zmene orientácie obrazovky sa nemenia/nevymieňajú- súradnicový systém sa nikdy nemení.
- Natívna poloha nemusí byť vždy taká, že zariadenie je orientované na výšku. Pri mnohých tabletových zariadeniach je prirodzená orientácia na šírku. Súradnicový systém senzorov je vždy založený na natívnej orientácii zariadenia.
- Ak aplikácia chce využiť údaje senzora a transformovať na súradnice displeja, musíme použiť metódu `getRotation()` na určenie rotácie obrazovky a následne použiť metódu `remapCoordinateSystem()` pomocou ktorej sa mapujú súradnice senzora na súradnice obrazovky.

> **Note**
>
> Niektoré senzory a metódy používajú súradnicový systém, ktorý je relatívny k svetovému referenčnému rámcu (na rozdiel od referenčného rámca zariadenia). Tieto senzory a metódy vracajú údaje, ktoré predstavujú pohyb zariadenia alebo polohu zariadenia vzhľadom na Zem. Pre viac informácií viď `getOrientation()` metóda, `getRotationMatrix()` metóda, senzor orientácie a senzor rotácie .

Osvedčené postupy pre prístup a používanie senzorov

- Senzory používajte iba keď je aplikácia v popredí
- od verzie Android 9 sa aplikácie, ktoré **bežia na pozadí** musia prispôsobiť nasledujúcim obmedzeniam:
  - V prípade nepretržitého vzorkovania, napr. senzor akceleromer, gyroskop, aplikácia nedostáva nové dáta.
  - V prípade jednorázového merania alebo udalosti o zmene meranej hodnoty, aplikácia taktiež nedostáva nové dáta.
- Vzhľadom na tieto obmedzenia je nutné používať senzor iba keď je aplikácia v popredí alebo využiť

`ForegroundService`

- Dbať na dodržiavanie životného cyklu aplikácie, t.j. `onPause()` odhlásiť príjem, udalostí od senzora.
- Testovanie senzorov pomocou **Android Emulatora**
- Neblokovať metódu `onSensorChanged()`
- Nepoužívajte metódy alebo typy snímačov, ktoré sú označené ako `deprecated` t.j. zastarané
- Pred použitím senzorov treba overiť či je senzor naozaj prítomný v zariadení
- Treba správne zvoliť vzorkovaciu frekvenciu - treba zabrániť zbytočnému plytvaniu CPU a energiou

Určenie polohy / Satelitná navigácia

Android poskytuje lokalizáciu na základe rôznych metód:

- triangulácia na základe BTS,
- WiFi prístupové body
- a najčastejšie GPS.

Cieľom tohoto riešenia je docieliť, čo najlepšie lokalizačné služby v rôznych podmienkach a obmedzeniach. Lokalizácia v exteriéry, v interiéri, malá spotreba energie, rýchla dostupnosť a požadovaná presnosť. To sú požiadavky, ktoré sú kladené na lokalizačné služby. Android OS má dva možné prístupy k takýmto lokalizačným službám.

1. `android.location` je API, ktoré poskytuje priamy prístup k lokalizačným službám. Tento prístup je nízko úrovňový a aby bola aplikácia optimalizovaná je potrebné, aby programátor mal dobré znalosti tejto problematiky. S každou novou verziou Android OS sa čiastočne menia podmienky, akým spôsobom sa k lokalizačným službám pristupuje a pri využití tohoto API je "na pleciach programátora" prispôsobiť aplikáciu aby fungovala aj na novšej verzii Andorid-u. **Pre všetky tieto nevýhody nie je odporúčané používať tento nízko úrovňový prístup**. Odporúčané je využívať prístup vyššej úrovne, ktorý je popísaný v nasledujúcej časti.
2. Rozhranie Google Location Services API , ktoré je súčasťou služieb Google Play, je preferovaným spôsobom, ako do aplikácie pridať lokalizačné služby. Ponúka jednoduchšie API, vyššiu presnosť, 'geofencing' s nízkou spotrebou a ďalšie.

Služba Google Location API

Táto služba je súčasťou rozsiahleho balíka Google play services [Location and Context APIs]. Tento balík obsahuje nasledujúce doplnkové služby:

Poloha:

- **Places API** - Poskytuje používateľom kontextové informácie o tom, kde sa nachádzajú a čo sa nachádza v ich okolí. Získate prístup k podrobným informáciám o 100 miliónoch miest v rôznych kategóriách.
- **Geofencing** - Geofencing kombinuje znalosť aktuálnej polohy používateľa a vedomosti o vzdialenosti používateľa k miestam, ktoré môžu byť zaujímavé.
  - Umožňuje nastaviť zemepisné hranice okolo konkrétnych miest a následne prijímať oznámenia, keď užívateľ vstúpi alebo opustí tieto oblasti.
  - Možnosť nastaviť viacero sledovaných oblastí.
  - Možnosť vytvorenia filtrov.
  - Optimalizovanie pre výdrž baterie: prispôsobenie frekvencie aktualizácií na základe polohy užívateľa a charakteru pohybu užívateľa (bez pohybu, chôdza, jazda autom, atď).
- **Location API** - Získajte údaje o polohe pre svoju aplikáciu na základe kombinovaných signálov zo senzorov zariadenia

pomocou rozhrania API účinného na batérie.

Činnosť

- **Platforma Google Fit** - záznam, spravovanie, vizualizácia fitness aktivít
- **API rozhranie na rozpoznávanie aktivity** - toto rozhranie spracováva signály z viacerých senzorov pričom dbá na nízku spotrebu zariadenia, aby presne detegovalo aktuálnu aktivitu používateľa.
- **Senzor API** - prístup k dátam zo senzorov

Čo je v okolí:

- **Správy z okolia**
- **Možnosť pripojiť sa na zariadenia v okolí**
- **Upozornenia z okolia** - služba zanikla v roku 2018

> **Použitie Location API vyžaduje Google Play services !**

Základné vlastnosti Location API:

Poskytovateľ polohy inteligentne riadi nízko úrovňové technológie a na základe požiadaviek na presnosť vracia čo najlepšie údaje o polohe. Požiadavky na presnosť sú určené intervalom "Vysoká presnosť" a "Nízka spotreba"

- Okamžitá dostupnosť
- aplikácia má okamžitý prístup k najlepšej aktuálnej polohe. Klasický proces získavania polohy môže trvať aj niekoľko desiatok sekúnd kým sa získa poloha s GPS systému. Okamžitá informácia o polohe je častokrát získaná z histórie polohy, ktorú si vyžiadala iná aplikácia. Z toho vyplýva, že presnosť tejto polohy nemusí byť dobrá, ale dá sa využiť na prvotné prispôsobenie kontextu aplikácie, ktorý sa po získaní presnejšej polohy znova upraví.
- Energetická efektívnosť
- minimalizovanie spotreby, na základe požiadaviek. Location API robí automatický výber najvhodnejších nastavení, pre dané požiadavky na presnosť.
- napr. aplikácia na popredí dostáva dáta z vysokou presnosťou, pri prechode aplikácie do pozadia sa presnosť obmedzí na periodické upozornenie na zmenu polohy.

Užitočné odkazy

- Nastavenie Google Play services
- Kontrola povolenia a získanie povolenia (permissions)

  > **Warning**
  >
  > Pri práci s mnohými API treba zistiť aké povolenia dané API vyžaduje. V starších verziách Android bolo postačujúce pridať dané povolenie do `AndroidManifest.xml` a užívateľ pri inštalácií potvrdil/zamietol pridelenie daného povolenia. Toto už, ale nie je postačujúce pri novších verziách Android OS, kde je už povinnosť si vyžiadať dané povolenie aj pri samotnom behu aplikácie, viď odkaz vyššie.

- Príklady na prácu s Location API:
  - Získanie poslednej známej pozície
  - Pravidelná aktualizácia polohy