



**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

BRNO UNIVERSITY OF TECHNOLOGY

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**

FACULTY OF INFORMATION TECHNOLOGY

**ÚSTAV INFORMAČNÍCH SYSTÉMŮ**

DEPARTMENT OF INFORMATION SYSTEMS

**KONTEJNER PRO MIGRUJÍCÍ SOFTWAREOVÉ KOM-  
PONENTY BEŽÍCÍ NA OS ANDROID**

A CONTAINER FOR MIGRATING SOFTWARE COMPONENTS RUNNING ON ANDROID OS

**DIPLOMOVÁ PRÁCE**

MASTER'S THESIS

**AUTOR PRÁCE**

AUTHOR

**Bc. VLADIMÍR ŠČEŠŇÁK**

**VEDOUCÍ PRÁCE**

SUPERVISOR

**RNDr. MAREK RYCHLÝ, Ph.D.**

**BRNO 2018**

## Abstrakt

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v českém (slovenském) jazyce.

## Abstract

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v anglickém jazyce.

## Klíčové slová

Sem budou zapsána jednotlivá klíčová slova v českém (slovenském) jazyce, oddělená čárkami.

## Keywords

Sem budou zapsána jednotlivá klíčová slova v anglickém jazyce, oddělená čárkami.

## Citácia

ŠČEŠŇÁK, Vladimír. *Kontejner pro migrující softwarové komponenty běžící na OS Android*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce RNDr. Marek Rychlý, Ph.D.

# Kontejner pro migrující softwarové komponenty bežící na OS Android

## Prehlásenie

Prehasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením RNDr. Mareka Rychlého, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....

Vladimír Ščešňák

27. marca 2018

## Podakovanie

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant, apod.).

# Obsah

<b>1</b>	<b>Úvod</b>	<b>2</b>
<b>2</b>	<b>Operačný systém Android</b>	<b>3</b>
2.1	Vrstvy operačného systému Android . . . . .	3
2.2	Aktivity a ich životný cyklus . . . . .	4
2.3	Služby (dlhotrvajúce operácie) v systéme Android. . . . .	6
2.4	Zabezpečenie . . . . .	8
2.5	Prístup k zdrojom . . . . .	10
<b>3</b>	<b>Použité technológie</b>	<b>13</b>
3.1	Serializácia . . . . .	13
3.2	Deserializácia . . . . .	13
3.3	Reflexia . . . . .	13
3.4	Wi-Fi Peer-to-Peer . . . . .	13
<b>4</b>	<b>Migrujúce softwarevé komponenty</b>	<b>14</b>
<b>5</b>	<b>Návrh kontajneru pre komponenty bežiacie na systéme Android</b>	<b>15</b>
5.1	Návrhový diagram . . . . .	15
5.2	Architektúra kontajneru . . . . .	15
<b>6</b>	<b>Implementácia</b>	<b>16</b>
<b>7</b>	<b>Tvorba vzorovej Android aplikácie</b>	<b>17</b>
<b>8</b>	<b>Záver</b>	<b>18</b>
	<b>Literatúra</b>	<b>19</b>

# Kapitola 1

## Úvod

*[TODO: Úvod, cieľ práce, teória, návrh...]*

*[TODO: kam by som mal dať reflexiu(?) - použitá technológia(?)]*

## Kapitola 2

# Operačný systém Android

Táto kapitola sa venuje operačnému systému Android, ktorý v roku 2007 uverejnilo konzorcium Open Handset Alliance, pozostávajúce z technologických spoločností ako je Google. Android bol uvedený pod open-source licenciou **Apache/MIT** a spolu s ním uzrelo svetlo sveta aj prvá beta verzia Android Software Development Kit (v skratke SDK). Ako uvádza [5] v priebehu pár mesiacov si stiahlo prvú beta verziu zo stránky Googlu niekoľko miliónov ľudí a v roku 2008 bol v Spojených štátoch amerických predstavený prvý mobilný telefón bežiaci na tejto platforme.

Cieľom uvedenia tohto operačného systému bolo zjednodušenie vývoja a predávania softvérových aplikáci bežiacich na mobilných telefónoch a tiež zaviesť štandard podobný PC a Macintoshu, ktorý by bol primárne určený pre mobilné telefóny. V súčasnosti sa ale táto platforma rozšírila aj na mikropočítače, inteligentné televízory (známe ako Android TV), inteligentné hodinky (Android Wear), palubné systémy aut a ďalšie druhy elektroniky. V čase písania tejto práce je operačný systém Android dostupný vo verzii 8.0 Oreo. Informácie uvedené v tejto kapitole boli čerpané zo zdroja [2]

### 2.1 Vrstvy operačného systému Android

Ako môžeme vidieť na obrázku 2.1 samotný Android je založený na **linuxovom jadre**, čož je aj jeho prvá vrstva. To umožňuje využiť kľúčové funkcie zabezpečenia a tiež umožňuje výrobcovi zariadení ľahko vyvinúť ovládače hardvéru.

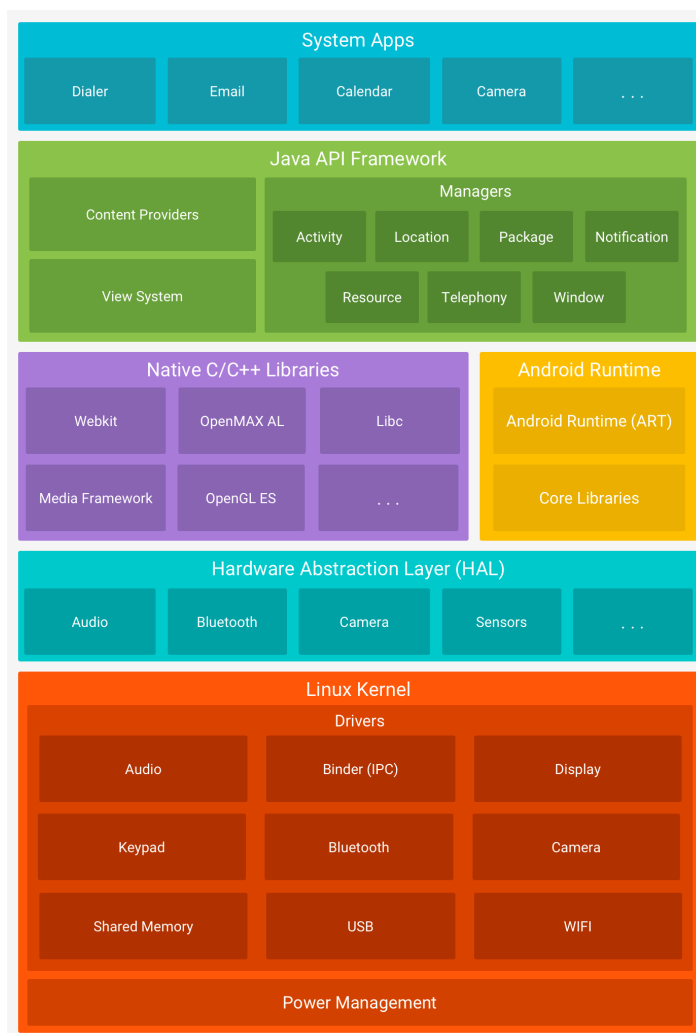
Ďalšou vrstvou je **hardvérová abstrakčná vrstva**, ktorá pozostáva z viacerých modulov a poskytuje štandardné rozhrania, ktoré vystavujú schopnosti hardvéru zariadenia do vyššej úrovni aplikačného rámca Java API. Úlohou vrstvy, ktorá je označená ako **Android Runtime** je tá, že každá aplikácia beží vo svojom vlastnom procese a má vlastnú inštanciu **Android Runtime**. Tá vykonáva beh viacerých virtuálnych strojov s nízkou pamäťou pomocou DEX súborov, čo je formát bytekódu špeciálne navrhnutého pre Android.

Mnoho základných komponentov a služieb systému Android, ako už aj vyššie spomenuté, vyžadujú natívne knižnice napísané v jazyku C a C++ a samotný Android poskytuje aplikačné rozhranie, pre jednoduchší prístup z vyvíjaných aplikácií.

**Java API Framework** obsahuje základné stavebné bloky, písané v jazyku Java, ktoré potrebuje programátor pri vytváraní aplikácií. Existujú štyri rôzne typy komponentov aplikácie a to Aktivity, Služby, Prijímače a Poskytovatelia obsahu. Hlavne vďaka týmto blokom, môžeme ľahko vytvárať prívetivé používateľské rozhranie, pristupovať k zdrojom ako

napríklad grafika, či iné súbory, alebo spravovať samotné aktivity. Túto vrstvu si popíšeme detailnejšie v ďalších častiach.

Poslednou vrstvou sú **Systémové aplikácie**, kde ide o množinu základných aplikácií, ktoré sú dodávané s každým Androidom. Ide hlavne o aplikácie ako telefonovanie, vytváranie textových správ, kamera, kalendár apodobne.



Obr. 2.1: Platforma Android

## 2.2 Aktivity a ich životný cyklus

Aktivity sú základným stavebným blokom aplikácii postavených na platforme Android. Sú akýmsi vstupným bodom medzi užívateľom a aplikáciou. Samotná aktivita poskytuje okno, kde aplikácia vykresľuje svoje užívateľské rozhranie, ktoré môže vyplniť celú obrazovku, môže byť menšie ako celá obrazovka, alebo dokonca môže sa objaviť v popredí iného okna.

Trieda `Activity` je na rozdiel od iných programovacích paradigiem, kde sú aplikácie spúšťané volaním metódy `main()`, iniciované systémom Android v inštancii `Activity` volaním špecifických metód, ktoré odpovedajú konkrétnym fázam svojho životného cyklu.

Pri interakcii užívateľa prechádzajú jednotlivé inštancie triedy **Activity** rôznymi stavmi. Týchto stavov je šesť a konkrétne sú to stavy *Created*, *Started*, *Resumed*, *Paused*, *Stopped*, *Destroyed*. Trieda **Activity** nám poskytuje niekoľko spätných volaní, ktoré dovoľujú aktivite vedieť, že sa nejako zmenil ich stav na základe ktorého, systém vytvára, zastavuje, obnovuje alebo ničí proces, v ktorom je daná aktivita umiestnená. Fázy životného cyklu a spätné volania, ktoré ich indikujú, si popíšeme nižšie a je ich možné tiež vyjadriť aj orientovaným grafom na obrázku 2.2.

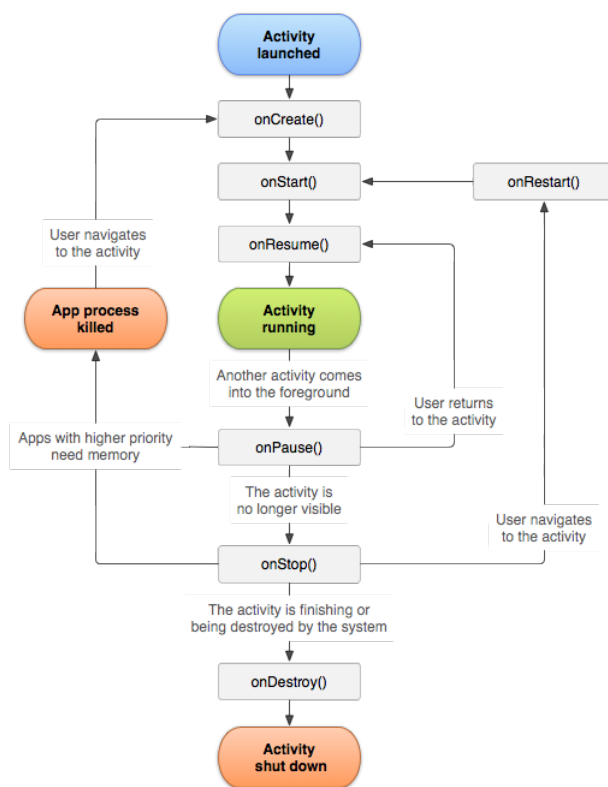
## Životný cyklus aktivity a jej spätné volania

- **onCreate()** - volanie, ktoré musí byť implementované. Toto volanie je spustené pri prvom vytvorení danej aktivity. V tomto bode vstupuje aktivita do stavu *Created*. V tejto metóde by mala byť implementovaná základná logika, ktorá sa vykoná len raz počas celej životnosti aktivity. Metóda obsahuje parameter **savedInstanceState** typu **Bundle**, obsahujúci predtým uložený stav aktivity. Ak je aktivita vytváraná prvýkrát, hodnota tohto parametru je **null**. Aktivita ale hneď po spustení tejto metódy prejde do stavu *Started* a systém rýchlo za sebou zavolá metódy **onStart()** a **onResume()**.
- **onStart()** - ak sa aktivita už nachádza v stave *Started*, systém vyvoláva toto spätné volanie. Metóda má na starosti to, že robí aktivitu viditeľnú pre používateľa a pripravuje ju na možnú interakciu. Na tomto mieste by sa mal inicializovať kód, ktorý udržiava používateľské rozhranie, alebo registrovať príjmač **BroadcastReceiver**, ktorý sleduje zmeny, na základe ktorých sa potom mení používateľské rozhranie. Po vykonaní tejto metódy aktivita prejde do ďalšieho stavu *Resumed* a volá metódu **onResume()**.
- **onResume()** - ak aktivita prejde do stavu *Resumed*, čo môže nastať buď prechodom zo stavu *Started*, alebo *Paused*, je vyvolaná táto metóda. V tomto stave aplikácia komunikuje s používateľom a reaguje na jeho interakciu, čo môže byť vybranie rôznych prvkov aplikácie, klepnutím, či iným povoleným gestom na dispeleji hardvéru, na ktorom tento operačný systém beží. V tejto metóde by sa mali inicializovať komponenty, ktoré neskôr v metóde **onPause()** uvoľníme. Aktivita a s ňou celá aplikácia ostáva v tomto stave, až kým nenastane udalosť podobná prejdenu užívateľa do inej aktivity, prijatia hovoru, či vypnutie obrazovky zariadenia, kedy daná aktivita stráca pozornosť. Ak dôjde k tejto udalosti, aktivita prejde do stavu *Paused* a vyvolá sa metóda **onPause()**.
- **onPause()** - systém zavolá túto metódu ako prvú indikáciu toho, že používateľ opúšťa danú aktivitu. Nie vždy to znamená, že užívateľ aplikáciu zruší a musí dôjsť k volaniu metódy **onDestroy()**. V tomto momente sa aktivita nachádza v stave *Paused* a v tejto metóde by mali byť implementované veci ako zastavenie rôznych animácií, prehrávanie médií, alebo uvoľnenie rôznych komponent. Vykonanie tejto metódy však tiež neznačí prechod do iného stavu. V tomto stave aktivita ostáva, dokiaľ nedôjde znovu k obnoveniu aktivite, v tomto prípade sa stav zmení na *Resumed* a systém vráti uloženú inštanciu danej aktivity. Ak je aktivita úplne neviditeľná, stav aktivity sa zmení na *Stopped* a systém zavolá metódu **onStop()**.
- **onStop()** - ak je už aktivita neviditeľná pre používateľa, vstupuje do stavu *Stopped*. To sa môže stať napríklad vtedy, ak je spustená nová aktivita, ktorá pokrýva celú obrazovku. Tu by sa mali začať uvoľňovať zdroje, ktoré používateľ ďalej nepotrebuje.



Tiež by malo dôjsť k uvoľneniu prijímača **BroadcastReceiver**, ak bol zaregistrovaný v metóde **onStart()**. Aby sa predišlo pretečeniu pamäti je dôležité v tomto mieste uvoľniť všetky prostriedky, ktoré programátor použil, nakoľko je možné, že systém zabije proces bez toho, aby zavolať poslednú metódu, životného cyklu aktivity, metódu **onDestroy()**. Z tohto stavu môže byť aktivita vrátená k interakcii s používateľom a systém zavolá metódu **onRestart()** a prejde do stavu *Started*, alebo dôjde k ukončeniu aktivity a systém zavolá metódu **onDestroy()** a prejde do stavu *Destroyed*.

- **onDestroy()** - metóda, ktorá je volaná pred zničením aktivity. Je to posledná metóda, ktorú aktivita dostáva a v tomto momente ju systém vyvolá, pretože bola vyvolaná metóda **finish()**, alebo systém ničí proces obsahujúci túto aktivitu kvôli šetreniu zdrojov. Táto metóda môže byť volaná aj v prípade, že dôjde k zmene orientácie obrazovky no hneď na to, je volaná metóda **onCreate()**, aby sa obnovil proces a s ňou aj komponenty, ktoré obsahuje a došlo tak k prekresleniu obrazovky. Táto metóda uvoľňuje všetky zdroje, ktoré neboli uvoľnené skoršími volaniami ako je napríklad metóda **onStop()**.



Obr. 2.2: Životný cyklus aktivity

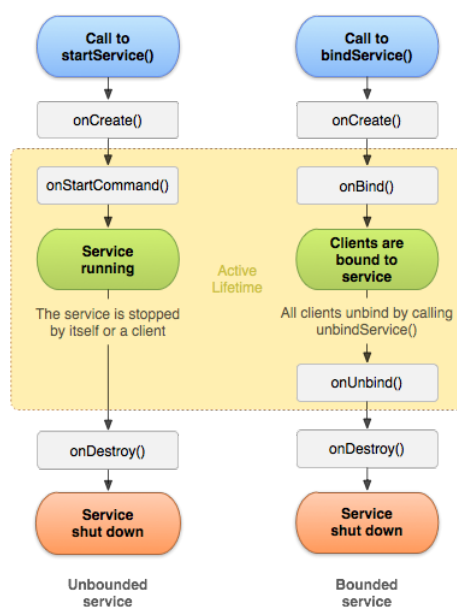
## 2.3 Služby (dlhotrvajúce operácie) v systéme Android.

Služba je komponenta aplikácie, ktorá má na starosti vykonávanie dlhodobých operácií na pozadí, bez užívateľského rozhrania. Službu môže spustiť iná komponenta, ako napríklad aplikácia, a táto služba beží na pozadí naďalej aj keď dôjde k prepnutiu na inú aplikáciu.

Služba môže z pozadia spracovávať sieťové transakcie, prehrávať hudbu, spracovávať vstupy/výstupy súborov, alebo komunikovať s poskytovateľom obsahu. Existujú 3 rôzne typy služieb:

- **Foreground** - ide o službu, ktorá sa vykonáva na popredí a je teda viditeľná pre používateľa a sú indikované zobrazením notifikácie v status bare. Ako príklad si môžeme uviesť prehrávanie hudby. Služby tohto typu pokračujú vo vykonávaní, aj keď používateľ nijako neinteraguje s aplikáciou.
- **Background** - služba, ktorá vykonáva operáciu na pozadí a používateľ ju tak priamo nezaznamenáva. Ide napríklad o získavanie aktuálnej polohy užívateľa.
- **Bound** - službu nazveme bound práve vtedy, ak sa k nej viaže nejaká komponenta (napríklad Aktivita), ktorá využíva volanie metódy `bindService()`. Táto služba ponúka rozhranie klient-server, ktorý umožňuje komponentám komunikovať so službou, posilať žiadosti, prijímať ich výsledky a komunikovať medzi procesmi. Služba beží len do doby, kým je k nej naviazaná iná komponenta, pri odviazaní sa služba zničí.

Ako môžeme vidieť na obrázku 2.3, životný cyklus služieb je jednoduchší ako vyššie životný cyklus aktivít.



Obr. 2.3: Životný cyklus služieb

Samotný životný cyklus služby, jeho vytvorenie až zničenie, môže vzniknúť z týchto dvoch možností

- **Spustenie služby** - pri tejto možnosti, je služba vytvorená, ak iná komponenta zavolá metódu `startService()`. Táto služba beží po dobu neurčitú a musí sa sama zastaviť vyvolaním metódy `stopSelf()`. Tiež ju môže zastaviť iná komponenta volaním metódy `stopService()`. V prípade, že je služba zastavená, systém ju zničí.
- **Viazaná služba** - druhá možnosť je tá, že služba je vytvorená keď iná komponenta (klient) volá metódu `bindService()`. Klient potom komunikuje so službou

prostredníctvom rozhoranie `IBinder`. Klient môže uzavrieť toto spojenie volaním metódy `unbindService()`. Na túto službu sa môžu viazať viacerí klienti, a ak sa všetci odpoja, je služba zničená systémom. Táto služba sa nemusí byť zastavovaná sama sebou.

Tieto dve možnosti nie sú úplne oddelené a môžeme sa naviazať aj na službu, ktorá bola spustená vyvolaním metódy `startService()`. Implementovaním týchto metód môžeme potom sledovať dve vnorené cykly životného cyklu služby. Prvý cyklus, ktorý môžeme pozorovať na obrázku 2.3 vľavo, sa sústreďuje na čas, medzi volaniami metód `onCreate()`, kde môže služba podobne ako aktivita inicializovať rôzne komponenty, alebo vytvárať nové vlákna, a metódou `onDestroy()`, kde dôjde k uvoľneniu všetkých použitých zdrojov. Druhý cyklus môžeme vidieť na rovnakom obrázku vpravo, kde aktívna životnosť služby začína volaním metódy `onStartCommand()`, alebo `onBind()`. Každá metóda je potom spracovávaná v zmysle `Intentu`, ktorý bol predávaný ako parameter metódam `startService()`, alebo `bindService()`. Ak je služba spustená, jej aktívna životnosť sa končí súčasne s ukončením celej životnosti a ak je služba viazaná, tak aktívna životnosť končí návratom z metódy `onUnbind()`.

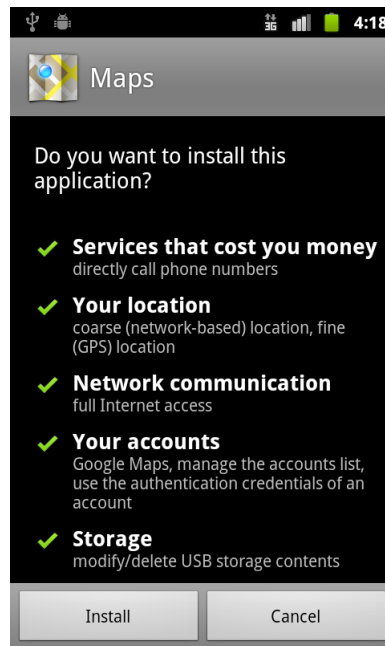
## 2.4 Zabezpečenie

Systém Android obsahuje prvotriedne bezpečnostné funkcie a spolupracuje s vývojarmi a implementátormi zariadení, aby udržali platformu Android a celý ekosystém bezpečný. Robustný a bezpečnostný model je nevyhnutný na to, aby bol zabezpečený silný, energický ekosystém pre aplikácie a zariadenia postavené na tejto platforme. Výsledkom toho je aj to, že celý životný cyklus vývoja Androidu podlieha prísnemu bezpečnostnému programu. Systém Android je navrhnutý tak, aby chránil dôvernosť, integritu a dostupnosť používateľov, dát, aplikácií, zariadenia a siete.

Android bol navrhnutý s viacvrstvovou bezpečnosťou, ktorá je dostatočne flexibilná na to, aby podporovala otvorenú platformu, akou je Android a zároveň chránila všetkých používateľov. Android ponúka programátorom aplikácii pomoc pri riešení bezpečnostných otázok a to vydávaním stabilnej platformy. Tiež existuje tím, ktorý sa o bezpečnosť stará a kontroluje aplikácie a ich potencionálne zraniteľné miesta a navrhuje programátorom spôsoby, ako tieto problémy riešiť. Systém Android nezabúda ani na používateľov a dovoľuje im chrániť svoje súkromie tým, že im zobrazuje oprávnenia, ktoré môže daná aplikácia vykonávať (obrázok 2.4), a ak sa používateľovi nepáči, môže oprávnenie zakázať.

Android beží na širokej škále hardvérových konfigurácií a hoci je procesorovo-agnostický (beží na rôznych procesoroch rovnako), využíva výhody určitých hardvérových bezpečnostných funkcií, ako je napríklad `ARM eXecute-Never`<sup>1</sup>. Samotný operačný systém Android je postavený na jadre linuxu. Všetky prostriedky zariadenia, ako sú napríklad funkcie fotoaparátu, údaje GPS, telefónne funkcie, sú prístupné cez operačný systém. Aplikácie určené pre platformu Android sú najčastejšie písané v programovacom jazyku Java a bežia v Android runtime (ART). Aplikácie bežia v rámci bezpečnostného prostredia, ktorý je obsiahnutý v rámci aplikačného sandboxu. Aplikácie tak získavajú osobitnú časť súborového systému, v ktorom môžu zapisovať súkromné údaje, vrátane databáz a nespracovaných súborov.

<sup>1</sup><http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0360f/CACHFICI.html>



Obr. 2.4: Príklad oprávnení, ktoré aplikácia požaduje

## Zabezpečenie na úrovni jadra

Základom pre mobilné počítačové prostredie je jadro linuxu s operačným systémom Android s niekoľkými kľúčovými bezpečnostnými funkciami, ktorými sú užívateľsky založený model oprávnení, izolovanie procesov, rozšírený mechanizmus pre bezpečnú medziprocesorovú komunikáciu a schopnosť odstrániť nepotrebné a potenciálne neisté časti jadra. Základným bezpečnostným cieľom jadra linuxu je navzájom izolovať medzi sebou zdroje používateľov. Linux tak zabráňuje používateľovi *A* čítať súbory používateľa *B*, zabezpečuje, aby používateľ *A* nevyčerpal pamäť používateľa *B*, zabezpečuje, aby používateľ nevyčerpal zdroje používateľa *B* a zabezpečuje, aby používateľ *A* nevyčerpal zariadenia používateľa *B* (napríklad GPS, Bluetooth).

## Aplikačný sandbox

Systém Android priradzuje každej aplikácii jedinečné používateľské ID (UID) a spúšťa ho ako samostatný proces. Tento prístup sa líši od ostatných operačných systémov, kde sa používajú viaceré aplikácie s rovnakými oprávneniami používateľa. Jadro vynucuje bezpečnosť medzi aplikáciami a systémom na úrovni procesov prostredníctvom štandardných vlastností linuxu, ako sú identifikátory používateľov a skupín, ktoré sú priradené aplikáciám. Ak sa napríklad aplikácia *A* pokúsi urobiť niečo škodlivé, ako je čítanie údajov aplikácie *B*, operačný systém to chráni, pretože aplikácia *A* nemá príslušné používateľské privilégia. Sandbox je teda jednoduchý, kontrolovateľný a založený na oddeľovaní procesov a oprávnení v štýle UNIX-u.

Vďaka tomu že je aplikačný sandbox v jadre, sa tento bezpečnostný model rozširuje aj na natívny kód a aplikácie operačného systému. Všetok softvér, ktorý beží nad jadrom, ako sú knižnice operačného systému, alebo aplikácie, bežia v aplikačnom sandboxe. V systéme Android neexistujú žiadne obmedzenia, ako písať aplikáciu, tak aby bola docieľená potrebná bezpečnosť a v tomto ohľade je natívny kód rovnako bezpečný ako aj interpretovaný kód.

Aby sa predošlo tomu, že dôjde k poškodeniu pamäte v jednej aplikácii a neohrozilo to ostatné aplikácie a bezpečnosť zariadenia, bežia aplikácie v aplikačnom sandboxe na úrovni operačného systému. Takto chyba v pamäti umožní ľubovoľné spustenie kódu len v kontexte konkrétnej aplikácie s povoleniami, ktoré vytvoril operačný systém. Treba však podotknúť, že tak ako všetky bezpečnostné funkcie, ani aplikačný sandbox nie je nezlomný.

## Kryptografia

Pre aplikácie Android tiež poskytuje sadu kryptografických API. Patria k nim implementácie štandardných a bežne používaných kryptografických primitív, ako sú AES, RSA, DSA, SHA. Rozhrania API sa však používajú aj na protokoly vyššej úrovne ako sú protokoly SSL a HTTPS. Vo verzii Android 4.0 bola predstavená trieda `KeyChain`, ktorá umožňuje aplikáciám používať úložisko systémových poverení pre súkromné kľúče a refazce certifikátov.

## Konfigurácia zabezpečenia siete

Táto funkcia umožňuje aplikáciám prispôbiť nastavenia zabezpečenia siete v bezpečnom deklaratívnom konfiguračnom súbore bez úpravy kódu aplikácie. Tieto nastavenia je možné nakonfigurovať pre konkrétne domény a pre konkrétnu aplikáciu. Kľúčové funkcie:

- **Vlastné nastavenie dôvernosti** - dochádza k modifikácii certifikačných autorít, ktoré aplikácia považuje za dôveryhodné. Napríklad dôvera konkrétnym certifikátom s vlastným podpisom, alebo obmedzenie súboru verejných certifikačných autorít, ktorým aplikácia verí.
- **Prepisy len pri ladení** - pre ladiace účely, kedy sa neohrozuje základňa používateľov, ktorí aplikáciu už používajú.
- **Obmedzenie návštevnosti** - chránenie aplikácie pred pripojeniami pomocou nešifrovaného HTTP protokolu namiesto protokolu HTTPS
- **Pripojenie certifikátu** - obmedzenie, aby aplikácia dôverovala len k pripojeným certifikátom.

## 2.5 Prístup k zdrojom

Okrem vyššie spomenutého nám operačný systém Android ponúka tiež možnosti pristupovať k zdrojom zariadenia. Týmto zdrojmi sú úložisko súborov, poloha zariadenia (známa ako GPS), kamera, senzory a rôzne ďalšie prvky konektivity, ktoré nám dovoľujú spájať sa s inými zariadeniami a patria medzi nich napríklad technológia Bluetooth, NFC, USB.

### Úložisko súborov

Android ponúka niekoľko možností ako je možné ukladať údaje aplikácii. Ich použitie sa líši od veľkosti, či viditeľnosti pre ostatné aplikácie. Poďme si bližšie popísať aké možnosti nám systém ponúka:

- **Interné úložisko súborov** - v predvolenom nastavení sú súbory uložené do interného ukladacieho priestoru a k týmto súborom nemôžu pristupovať ostatné aplikácie. Pri odinštalovaní aplikácie sú súbory nachádzajúce sa v tomto úložisku odstránené.

Android nám tiež ponúka možnosť uchovávať niektoré údaje len dočasne, k čomu slúži špeciálny adresár vyrovnávacej pamäte, ktorý má každá aplikácia, avšak pri zaplnení interného úložiska nám tieto súbory systém zmaže. Pri odstránení sa tieto súbory taktiež odstraňujú.

- **Externé úložisko súborov** - tento úložný priestor je nazvaný externým, pretože prístup k nemu nie je zaručený. Používatelia môžu tento priestor pripojiť k počítaču ako externé pamäťové zariadenie a môže byť aj fyzicky odstrániteľné (SD karta). Súbory uložené na externom úložisku sú čitateľné pre všetkých používateľov a môžu byť modifikované. Toto úložisko by malo slúžiť na ukladanie používateľských údajov, ktoré by mali byť prístupné iným aplikáciám a mali by ostať uložené aj v prípade, že užívateľ aplikáciu odinštaluje. Súbory je možné ukladať aj do adresáru špecifického pre danú aplikáciu, najmä ak ide o veľké dáta a tieto údaje sa pri odinštalovaní aplikácie odstraňujú.
- **Zdieľané preferencie** - toto riešenie je vhodné, ak nie je potrebné ukladať množstvo informácií a nevyžaduje sa ani žiadna štruktúra. Ide vlastne o ukladanie primitívnych dát vo formáte kľúč-hodnota. Pre zjednodušenie práce s týmito primitívami existuje rozhranie `SharedPreferences`, ktoré uľahčuje ukladanie týchto typov. Toto rozhranie ukladá páry kľúč-hodnota do súborov XML, ktoré pretrvávajú v reláciách používateľov, aj keď je aplikácia zabitá. Úložisko je vhodné pre ukladanie jednoduchých dátových typov a údajov, ako je napríklad uloženie najvyššieho skóre užívateľa.
- **Databázy** - Android poskytuje plnú podporu `SQLite` databázam. Pri vytváraní databázy, je databáza prístupná len aplikácii, ktorá ju vytvorila a je vhodná pre ukladanie štrukturovaných údajov.

## Prístup k poloha zariadenia

Jednou z jedinečných funkcií mobilných aplikácií je informovanosť o polohe. Používatelia mobilných zariadení si so sebou berú svoje zariadenia a pridávanie informácií o lokalite do aplikácie ponúka používateľom viac kontextuálny zážitok. Rozhranie pre poskytovanie lokality umožňujú do aplikácie pridávať informáciu o polohe pomocou automatického sledovania polohy, geofencingu a rozpoznávaní činností.

## Prístup ku kamere zariadenia

*[TODO: zvážiť, či je potrebné]*

Systém obsahuje plnú podporu pre rôzne fotoaparáty a ich funkcie, ktoré sú v zariadeniach k dispozícii. Umožňuje tak vývojárom aplikácii zachytávať fotky, či videá.

## Snímače

Väčšina zariadení so systémom Android má vstavané snímače, ktoré merajú pohyb, orientáciu a rôzne podmienky prostredia. Tieto snímače sú schopné poskytovať hrubé dáta s vysokou presnosťou. Sú užitočné pri polohovaní trojrozmerného zariadenia, alebo pri sledovaní zmien okolitého prostredia. Niektoré snímače sú hardvérové, iné zas softvérovo založené. Android podporuje tri široké kategórie senzorov:

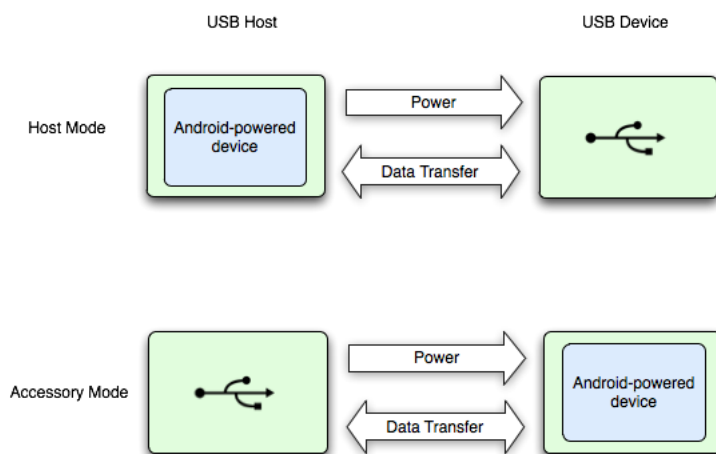
- **Snímače pohybu** - snímače, ktoré merajú zrýchlenie a silu otáčania pozdĺž troch osí. Táto kategória zahŕňa senzory ako akcelometre, gravitačné senzory, gyroskopy a rotačné vektorové snímače.
- **Snímače prostredia** - tieto snímače merajú rôzne enviromentálne parametre, ako je teplota, tlak okolitého vzduchu, osvetlenie a vlhkosť. Do tejto kategórie spadajú barometre, fotometre a teplomery.
- **Snímače polohy** - merajú fyzickú polohu zariadenia. Tu patria senzory orientácie a magnometre.

## Prvky konektivity

Podpora technológie **Bluetooth** nie je výnimkou ani u tohto operačného systému. Táto technológia umožňuje zariadeniam bezdrôtovo vymieňať dáta s inými Bluetooth zariadeniami. Android ponúka aplikačný rámec pre prácu s Bluetooth funkciami. Prostredníctvom tohto rámcu môžu aplikácie vykonávať skenovanie ďalších Bluetooth zariadení, pripojenie sa k iným zariadeniam prostredníctvom vyhľadávania služby, preniesť údaje do a z iných zariadení.

Ďalšiou podporovanou je služba **Near Field Communication** skôr známa pod skratkou NFC. Služba umožňuje zdieľať malé dáta medzi štítkami NFC a zariadením, ktoré túto technológiu podporuje, alebo medzi dvoma takými to zariadeniami. Existujú jednoduché štítky, ktoré ponúkajú len sématické čítanie a písanie a niekedy umožňujú len čítanie karty. Na druhej strane existujú aj zložité štítky, ktoré ponúkajú rôzne matematické operácie a obsahujú kryptografický hardvér, na overovanie prístupu.

Android podporuje aj rôzne periférie **USB** a to pomocou dvoch režimov a to USB príslušenstvo (USB accessory) a USB hostiteľ (USB host), ktoré sú znázornené aj na obrázku 2.5. V prvom menovaní funguje externý USB hardvér ako hostiteľ a ako príklad si môžeme uviesť dokovacu stanicu. V hostiteľskom režime funguje zariadenie ako hostiteľ a medzi príklady patria napríklad digitálne kamery.



Obr. 2.5: USB režimy

## Kapitola 3

# Použité technológie

*[TODO: Serializácia, deserializácia, reflexia]*

[1] a [4]

### 3.1 Serializácia

*[TODO: Vysvetliť]*

### 3.2 Deserializácia

*[TODO: Vysvetliť]*

### 3.3 Reflexia

*[TODO: vysvetliť]*

[3]

### 3.4 Wi-Fi Peer-to-Peer

*[TODO: co to je, ako to funguje, na co to budem pouzivat]*



## Kapitola 4

# Migrujúce softwarevé komponenty

*[TODO: čo to je, existujúce riešenia]*

## Kapitola 5

# Návrh kontajneru pre komponenty bežiacie na systéme Android

*[TODO: nejaké základne myšlienky, čo a prečo]*

### 5.1 Návrhový diagram

### 5.2 Architektúra kontajneru

*[TODO: uml...popísať, čo je čo]*

## Kapitola 6

# Implementácia

*[TODO: popísať len kontajner, alebo spojiť to aj s android aplikaciou?]*

## Kapitola 7

# Tvorba vzorovej Android aplikácie

*[TODO: všetko k tomu]*

## Kapitola 8

## Záver

*[TODO: čo sa podarilo, čo nie, aký je výsledok....]*

# Literatúra

- [1] Bloch, J.: *Effective Java*. Addison-Wesley, třetí vydání, 2018, ISBN 0134685997, 9780134685991.
- [2] Developers, A.: *Android developer guide*. [Online; navštíveno 15.03.2018].  
URL <https://developer.android.com/guide/index.html>
- [3] Forman, I. R.; Forman, N.: *Java Reflection in Action (In Action Series)*. Greenwich, CT, USA: Manning Publications Co., 2004, ISBN 1932394184.
- [4] Genčúr, M.: *Rámec pro dynamickou aktualizaci aplikací v jazyce Java*. [Online; navštíveno 15.03.2018].  
URL <http://www.fit.vutbr.cz/study/DP/DP.php?id=6955&file=t>
- [5] Rogers, R.; Lombardo, J.; Mednieks, Z.; aj.: *Android Application Development: Programming with the Google SDK*. O'Reilly Media, Inc., první vydání, 2009, ISBN 0596521472, 9780596521479.