Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Vývoj mobilní aplikace pro Android na testování internetového připojení

Diplomová práce

Autor: Josef Jakub Jestřáb
Studijní obor: K-AI-2

Vedoucí práce: doc. Ing. Ph.D. Filip Malý

Hradec Králové červen 2015

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne Josef Jakub Jestřáb

Poděkování

Děkuji svému vedoucímu práce za vybrání velmi zajímavého tématu diplomové práce a za její ukázkové vedení. Bez jeho podpory by pro mě bylo obtížné studium dokončit.

Anotace práce

Tato práce obsahuje teoretické a praktické zkušenosti z oblasti vývoje mobilních technologií, za pomoci vlastního použití programovacího jazyka Java v prostředí Android Studia. Taktéž jsou zde zahrnuty způsoby měření rychlosti internetového připojení a další návrhy případného rozvoje a obecná doporučení při vývoji takového programu. Práce je svým zacílením určena spíše nepříliš zkušeným programátorům mobilního operačního systému Android.

Z toho důvodu se i v úvodní části věnuje obecnému vývoji mobilní aplikace a představuje tak nezbytný základ pro práci v tomto prostředí. Zmiňuje nejen rámcový přehled, ale i specifika, která se vztahují k vývoji v Androidu pro mobilní zařízení. Na tuto část navazuje kapitola zabývající se možnostmi měření internetového připojení z technického i praktického aspektu. K získání relevantních výsledků s ohledem na akceptovatelné uživatelské chování je zde uvedeno více metod, z nichž je následně vybrána jedna, která bude použita v praktické části. Tato kapitola obsahuje popis reálného projektu, na němž je vysvětlen samotný vývoj aplikace včetně specifických situací. Pro demonstraci jsou zde uvedeny příklady vývoje kódu aplikace určené k měření rychlosti internetu.

Cílem této práce je čtenáři osvětlit podstatu, způsoby, komplikace a vybrané postupy vývoje pro prostředí Android a na konkrétním příkladu poukázat na charakteristiky spojené s takovou činností. Práce se zabývá možnostmi získání relevantních dat z oblasti měření rychlosti internetu a nabízí zamyšlení nad dalšími možnostmi rozšíření.

Annotation

Title: Development of Android application for testing of internet connection

This Diploma Thesis contains theoretical and practical knowledge of language Java in Android Studio interface. Also they are added ways of internet connection measurement, suggestions for additonal upgrade and best practices in development of this kind of programme. This work suppose to enlighten begginers with mobile applications.

Because of it, first part is related to basics of development for Android and introduce necessary base for work in such environment. There is mentioned not only general view, but also you can find there specific entities connected with development of mobile applications and Android. On this part continues chapter about internet connection measurement form technical and practical point of view. It involves more methods of how to recieve relevant results in scope with user friendly behaviour and from this was chosen one method, which was used in practical part. There is description of real project, on which is explained process of development and specific situations connected with it. It includes examples with the code for demonstration of whole experience of development of application for testing of internet connection.

Objective of this Diploma Thesis is to explain to reader the base, ways, complications and practices of development for Android environment and on specific example shows the characteristics connected with such activity. It involves options how to recieve relevant data from speed measurement and suggests some possible upgrades.

Obsah

Úvod 1

1. Základy mobilní aplikace 1

1.1 Charakteristické vlastnosti mobilních aplikací 3

1.2 Základní struktura 4

1.3 Zobrazení 7

1.4 Další typy komponent 10

1.5 Struktura 11

1.6 Nastavení 13

1.7 Uživatelské rozhraní 15

1.8 Komunikace s okolím 16

1.9 Upload 17

2. Měření připojení 18

2.1 Hardware 18

2.2 Problematika měření 19

2.3 Výběr metody 22

3. Praktická část 23

3.1 Zadání 23

3.2 Dokumentace 24

3.3 Prvotní nastavení 24

3.4 Rozvržení 24

3.5 Manifest 26

3.6 První obrazovka 27

3.7 Test připojení 32

3.8 Měření stahování 34

3.9 Měření odesílání 38

3.9.1 Případ 1 38
3.9.2 Případ 2 41
3.9.3 Serverová část 43
3.10 Odeslání hodnot 43
3.11 Dokončení aplikace 44
3.12 Závěr praktické části 46
Shrnutí výsledků 47
Závěry a doporučení 48
Seznam literatury 49
Přílohy 50
Seznam obrázků 50
Seznam tabulek 51
Seznam ukázek 51
Zadání práce 53

Úvod

Programování pro mobilní zařízení je velmi specifická činnost. Android používá vlastní metody využití hardwarových i softwarových prostředků s přihlédnutím na jejich omezenost a variabilitu napříč jednotlivými modely a značkami zařízení. Tato práce si klade za úkol na konkrétním příkladu představit základní principy vývoje.

Při realizaci samotného projektu vyvstalo více otázek a nuancí, jakými se lze zabývat. Mezi ty základní patří:

- Jak se bude aplikace chovat.
- Jak zajistit co největší přesnost měření.
- Jak zajistit konzistentní korektnost procesu i při vícenásobném simultánním použití.
- Jak zajistit dostačující uživatelskou přívětivost.
- Jak příliš nezatížit prostředky zařízení.
- Jaké výrazové prostředky bude aplikace využívat pro informování uživatele o stavu a výsledku měření.
- Jak udržet kód čitelný a zároveň dobře vysvětlitelný.

Další otázky nepochybně vyplynou při bližším ponoření do tématu. Na výše zmíněné okruhy práce budou v některých případech reagovat již teoretické části, zbylá problematika se promítne až v části praktické.

1. Základy mobilní aplikace

Aplikace pro mobilní zařízení jsou nyní velmi populárním nástrojem pro firmy, ať už jako prostředek reklamní kampaně nebo jako výdělečná činnost vývojových týmů v projektech, které mohou při vhodném nápadu vydělat velmi zajímavý finanční obnos.

Nezřídka se lze u úspěšných titulů setkat s počty stažení aplikací v řádu statisíců i milionů (1). K tomu dochází nejčastěji u aplikací pro operační systém Android, který je v tuto chvíli nejrozšířenějším operačním systémem na světě.[1]

Number of installs	Number of applications
1,000,000,000 - 5,000,000,000	12
500,000,000 - 1,000,000,000	10
100,000,000 - 500,000,000	98
50,000,000 - 100,000,000	118
10,000,000 - 50,000,000	1499
5,000,000 - 10,000,000	1999
1,000,000 - 5,000,000	13949
500,000 - 1,000,000	15006
100,000 - 500,000	82062

Tabulka 1: Počet Android aplikací dle počtu stažení (1)

Source	Year	Android	iOS/OS X	Windows	Others
Gartner	2014	48.61%	11.04%	14.0%	26.34%
Gartner	2013	38.51%	10.12%	13.98%	37.41%
Gartner	2012	22.8%	9.6%	15.62%	51.98%

Tabulka 2: Rošířenost operačních systémů na světě (2)

Z těchto statistik vyplývá, že čistě z hlediska jednoduchého rozšíření jsou aplikace pro operační systém Android nejzajímavější možností. Většina mobilních zařízení, od mobilů přes tablety, pracují v tomto prostředí. I Google play - služba pro instalaci programů v Androidu - je obsažena ve všech těchto zařízeních, což zajišťuje bezproblémovou distribuci. To ještě více zjednodušuje situaci případným projektům.

Proto je vývoj mobilních aplikací pro Android na velkém vzestupu a čím dál více vývojových týmů upíná svoji pozornost tímto směrem. Tato práce na zmíněný trend reaguje a snaží se přiblížit obecné principy programování konkrétně v jazyce Java a ve

[1] Zdroj Gartner inc. 2015 (2)

stále více populárním vývojovém prostředí Android Studio, které se specializuje právě na Android.

Aby byly takové principy prakticky doložitelné, byl pro tuto práci naprogramován jednoduchý pomocník pro zjištění rychlosti internetového připojení mobilního zařízení. Tím se prakticky prověří nejen základy programování v daném prostředí, ale také charakteristické situace, které se vztahují k mobilním zařízením a práci s jejich hardwarovými prostředky. Budou zde použity metody pro komunikaci s okolím a pro přijatelné zobrazení v různých typech zařízení.

V neposlední řadě zde bude také uvedeno, jaké postupy lze použít pro měření rychlosti a jaké aspekty se vztahují k různým metodám zjišťování rychlosti. Než se tak ale stane, je na místě zmínit základy samotného programování pro mobilní aplikace.

1.1 Charakteristické vlastnosti mobilních aplikací

Po přeskočení obecných základů programovacího jazyka Java, které jsou nezbytnou součástí znalostí, je třeba pochopit, že vývoj mobilních aplikací používá vlastní metody a funkce pro využívání prostředků zařízení. Je to dáno jednak operačním systémem, ale především samotnými hardwarovými možnostmi. Kromě nižší paměti RAM a menšího úložiště (viz podkapitola 2.1 Hardware) komplikuje vývoj také předpoklad, že více aplikací bude pravděpodobně fungovat jako služba[2], popř. funguje na pozadí, protože se uživatel věnuje jiné aplikaci nebo mobil není aktivně využíván v danou chvíli. V obou případech ale aplikace spotřebovávají hardwarové prostředky a tím snižují výkon telefonu. Velmi podstatnou veličinou je bezesporu spotřeba energie, neboť ta je také velmi kritická pro běžné používání. Proto je v případě náročných aplikací optimalizace kódu esenciální.

Dalším charakteristickým prvkem pro mobilní přístroje jsou různé úhlopříčky, rozlišení zobrazení a tedy hustota pixelů. Malé chytré telefony se pohybují od např. 400

[2] Služba = program bez grafického rozhraní, který např. přehrává hudbu na pozadí.

x 800 pixelů a necelých 3 palců až po tablety 2048 x 1536 a úhlopříčkou 10 palců. Aplikaci je tedy vhodné psát tak, aby se obraz přizpůsobil daným možnostem, což je blíže popsáno v podkapitole 1.3 Zobrazení.

V neposlední řadě se programování pro Android liší tím, že mobilní přístroj používá pro připojení se k různým zařízením specificky vytvořené metody a funkce ke komunikaci s jednotlivými prvky zařízení. Lze tedy nadneseně konstatovat, že Android pro jazyk Java zároveň představuje i framework[3] - účelně navržený pro programování na tuto specializaci. Stejně jako klasický framework používá Android svůj návrhový vzor a své metody pro obsluhu stavu aplikace obohacené o programové vybavení spojené s charakteristickými funkcionalitami jako práce s mobilní konektivitou různých druhů, s kontrolkami a tlačítky zařízení ad.

Pro pochopení frameworku a správné programátorské návyky je třeba začít od struktury aplikace.

1.2 Základní struktura

Stejně jako bylo přirovnáno Android prostředí k frameworku, lze přirovnat návrhový vzor mobilní aplikace k MVC[4], tak jako to udělal i Matěj Konečný ve svém známém seriálu o Androidu, publikovaném na serveru Zdroják.cz (3). Ten zmiňuje základní prvky aplikace **Activity** a **View**.

Activity představuje vzdáleně kontrolér – hlavní část programu, která zajišťuje veškeré fungování a zpracovávání. Kontrolér volá funkce pro získání dat a po získání je zpracovává. Od kontroléru se liší Aktivita[5] tak, že je i prezentační vrstvou aplikace.

View je iniciováno prostřednictvím Activity a má úlohu pouze v samotném zobrazení. Dle analogie s MVC, view je již transformované view do html podoby.

---

[3] Framework = ucelený balík funkcí a metod pro často řešené a nízkoúrovňové úlohy, nadstavba pro programovací jazyk pro ulehčení práce od rutinních a často se opakujících úkonů.

[4] Model View Controller = nejen webový návrhový vzor, kde model na popud kontroleru získává data z databáze nebo jiného persistentního úložiště (pakliže je potřeba dat), data zpracuje kontroler a pak již připravená data / obsah posílá do view, které je jen zobrazí.

[5] Pro rozlišení názvu komponenty Aktivita a podst. jména aktivita tato práce rozlišuje první písmena

Analogicky lze mít i databázi např. pomocí SQLite[6] a tím ukládat i data. To není tak častým prvkem u mobilních aplikací jako u webových projektů, neboť charakter projektů to málokdy vyžaduje a databáze zabírá permanentní místo telefonu. Obecně zařízení nedisponují velkými kapacitami, a proto mimo případů nezbytostnosti není databáze žádoucí.

Aktivita (Activity) je základním stavebním prvkem. Platí také, že co jedna obrazovka aplikace, to jedna Aktivita, a jedna Aktivita volá druhou. V případě, kdy další činnost nemusí být spojená s novou obrazovkou, jsou tu jiné možnosti popsané v podkapitole 1.4 Další typy komponent. Je nutné si uvědomit, že tím, že se otevírají Aktivity postupně, při ukončení aplikace nestačí zavřít jen aktuálně používanou Aktivitu, ale je třeba uzavřít hierarchicky i veškeré předchozí Aktivity včetně spouštěcí.

Volání další komponenty se provádí skrze tzv. **Intent**. Nelze tedy volat novou Aktivitu jako při běžném programování v jazyku Java (new NewActivityClass), nýbrž se zavolá nový Záměr (Intent). Ten mimochodem, kromě samotného zavolání, dokáže předat nové komponentě obsah proměnných, podobně jako je naznačeno v ukázce.

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.putExtra(Intent.EXTRA_EMAIL, recipientArray);
startActivity(intent);
```

Ukázka 1 – Volání intentu (4)

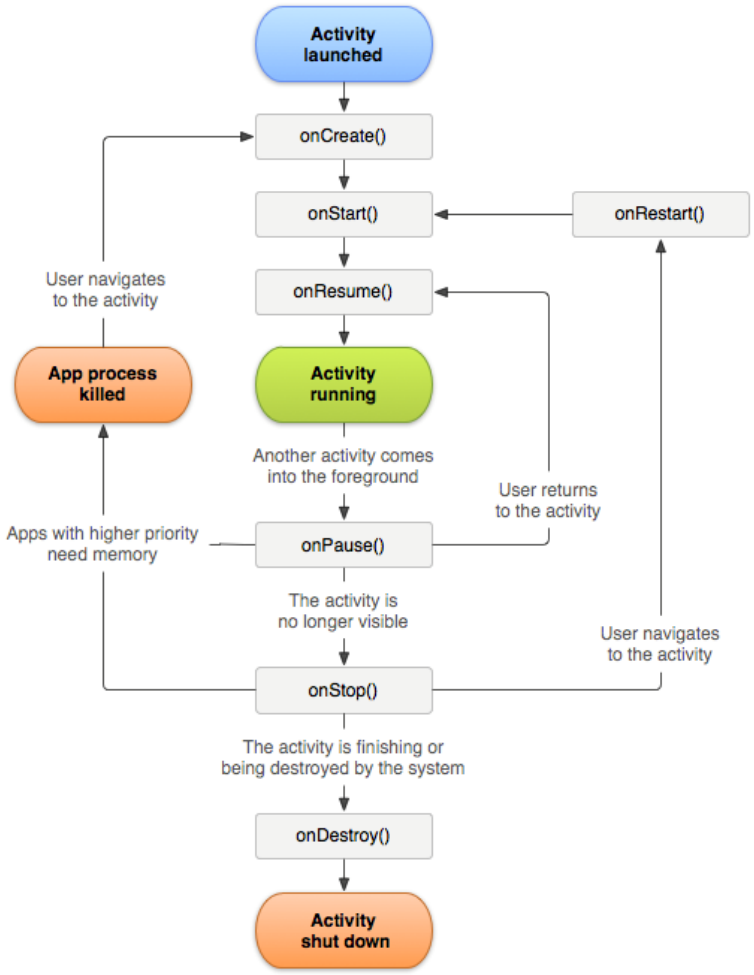
Tím se Intent stává ideální volbou i pro použití v případě posílání zprávy popř. dalšího obsahu nějaké jiné aplikaci např. pro otevření nějakého odkazu přes zvolený prohlížeč[7]. Intent využije pro volání metody `startService()` nebo `startIntent()`, což se váže k další specifické vlastnosti Aktivity a tou je její životní cyklus.

Ten není tak jednoduchý, jak by se mohlo zdát, ale při bližším zkoumání činností spojených s Aktivitou to logicky vyplývá. Spuštěnou aplikaci lze pozastavit, schovat ji

[6] SQlite je opensource SQL databázové řešení pro Android.

[7] Takovému Záměru se říká implicitní, protože nevolá konkrétní třídu, spíše volá spouštěč dané akce v aktuálním zařízení, který se o akce daného typu standardně stará. V případě, že je více možností a žádná nezvolena jako daná, dá Android uživateli na výběr z možných programů.

na pozadí a pracovat s jinou aplikací, poté se k původní vrátit. Následně může aplikace čekat na nějaká data z internetu a samozřejmě na konci je třeba Aktivitu a program zavřít. Tím Aktivita nabývá různých stavů, proto je při vývoji třeba ošetřit možnosti, jak se bude program chovat v případě podobného scénáře. Aplikace by rozhodně neměla zbytečně blokovat zdroje, jichž by mohla využít nově puštěná aktivita, stejně tak je třeba dbát na to, aby se i ve fázích mimo samotný běh aktivity (Activity running) aplikace stále chovala stabilně a šlo se do ní případně bez potíží vrátit atd.



Obrázek 1: Graf životního cyklu Aktivity (5)

Z logiky předchozích vět vyplývá, že Aktivitu lze pozastavit jiným procesem např. jinou aplikací uživatele. Nicméně, jak bylo řečeno výše, jedna Aktivita volá druhou, a proto stejně jako cizí aplikace i vnitřní Aktivita pozastaví původní Aktivitu. Aplikace, jež má hodně Aktivit, které volají jedna druhou, není z důvodu plýtvání prostředků nejideálnějším řešením. Proto by se při návrhu aplikace rozhodně mělo myslet

na to, jak se vyvarovat příliš mnoha zbytečně spuštěným Aktivitám a nepřetěžovat tak paměť. Pararelně je dobré se vyvarovat zbytečným pohledům a obrazovkám navíc.

1.3 Zobrazení

Jak již bylo zmíněno, jednou z komplikací vývoje aplikací pro mobilní přístroje je různorodost zobrazovacích ploch těchto zařízení. Jak uvádí například server svetandroida.cz (6), na viditelnost, rozložení a proporce prvků obrazovky se vztahuje více faktorů:

1. Šířka displeje ovlivňuje velikost zobrazovací plochy pro grafiku.
2. Rozlišení a fyzická hustota pixelů ovlivňují velikost grafických prvků.
3. Technologie dotyku může ovlivnit velikost a umístění některých prvků.
4. Verze OS Android ovlivňuje velikost a vzhled nativních prvků.
5. Režim orientace ovlivňuje počet a velikosti jednotlivých prvků.
6. Výbava zařízení ovlivňuje použitelnost a tedy výskyt grafických prvků.

Při vývoji aplikace se zohledňuje poměrně hodně možností a omezení. Některá se řeší při startu projektu, kde je třeba nastavit, jaké verze Androidu jím budou podporované. Další faktory, jako technologie dotyku a výbava zařízení, se řeší v průběhu návrhu jednotlivých view. Při samotném návrhu zobrazení má na prvky největší vliv šířka displeje, rozlišení, fyzická hustota pixelů a režim orientace, kde obecná řešení pracují především s těmito variantami:

1. Určit, jaké z view se má zobrazit pro jaký typ přístroje. V extrémních případech je možnost dokonce určit, že aplikace je vhodná jen pro daný jeden typ zařízení, nicméně pro standardní aplikace není rozumné se kvůli takové malé technické obtíži ochuzovat o potenciální množství zákazníků.
2. Obrazovku přepočítat tak, aby se prvky chovaly podobně jak na velkém, tak na malém zařízení.

Pro bližší popis je potřeba podrobněji popsat veličiny zmíněných vlastností:

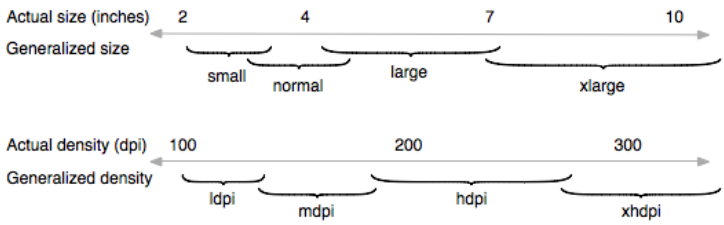
- **Šířka obrazovky** – měří se v palcích (1 palec rovná se 25,4 mm)
- **Rozlišení obrazovky** – standardně se měří v px

- **Hustota obrazovky** – alternativní možnost pro změření rozlišení, měří se v **dpi**[8], aby lépe vypovídala nejen o rozlišení displeje v počtu bodů, ale také o jejich rozložení a reálné velikosti případné grafiky. Jako alternativi k dpi jsou pak jednotky dle velikosti zařízení – ldpi, mdpi, hdpi a xhdpi.

Vztah dpi k dalším jednotkám je pak následovný:

- ldpi (100-120 dpi)
- mdpi (120-160 dpi)
- hdpi (160-240 dpi)
- xhdpi (240-320 dpi)

Aby mohl vývojář tuto rozmanitost[9] efektivněji řešit, nabízí Android zjednodušení ve formě nastavení skupin zařízení do 4 typů, viz obr.



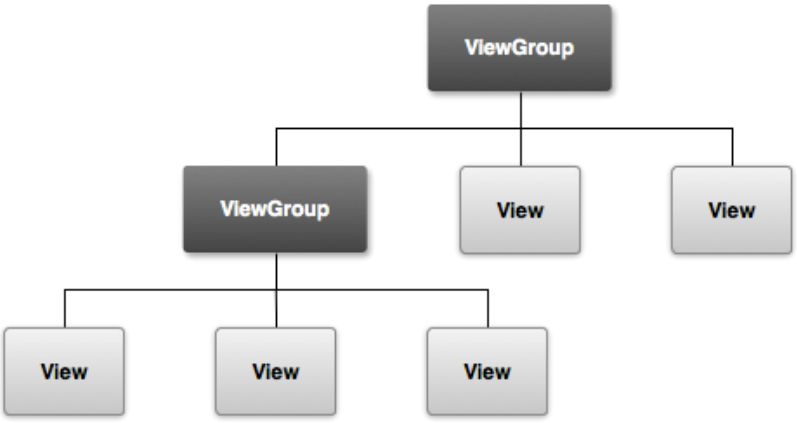
Obrázek 2: Rozdělení mobilních přístrojů podle zobrazení na 4 typy (7)

Tím se nabízí pohodlná možnost použití prvního obecného řešení, kdy lze díky jednoduché direktivě nastavit pro každý typ speciálně vytvořené view a tím se elegantně postarat o adekvátní zobrazení prvků skrze velkou škálu zařízení. V případě druhé varianty se situace také zjednodušila, neboť díky nastavení velikosti prvku v jednotkách dpi se prvek automaticky zmenšuje podle fyzické hustoty pixelů.

Další možností, jak ovlivnit zobrazení, jsou různé varianty **layoutu** a skládání **view**. Díky většímu množství prvků k zobrazení, lze jednu obrazovku rozdělit na více částí. Díky **ViewGroup**, což je technicky vzato jen speciální případ view, lze použít více prvků uvnitř jedné obrazovky, jak napovídá obrázek.

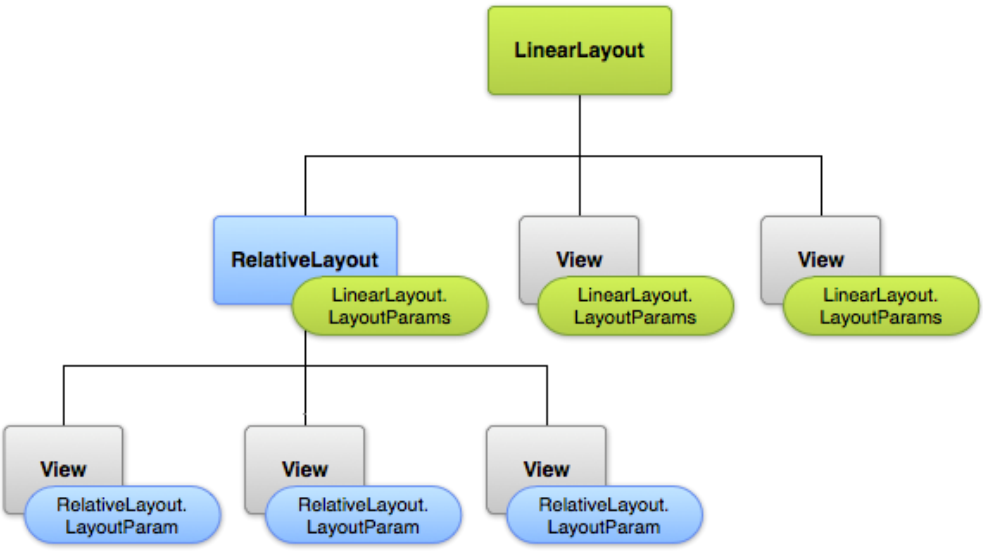
[8] DPI = Dots per inch – počet bodů které se vejdou na jeden palec

[9] kterou lze ještě výrazně zpodrobnit a prohloubit



Obrázek 3: Příklad struktury více ViewGroup a View (7)

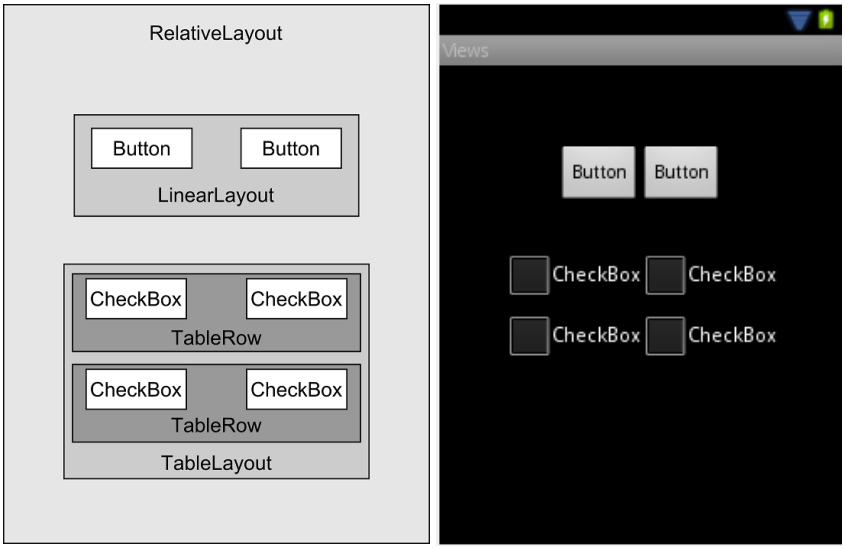
Pokud by struktura vypadala dle obrázku výše, jednotlivé prvky by byly na obrazovce rozložené bez řádu. Možnosti rozložení totiž určuje **layout**. Android má k dispozici sadu layoutů, které zpracovávají nejběžnější požadavky.



Obrázek 4: Vizualizace možné hierarchie s layout parametry (4)

Pro potřeby běžné aplikační obrazovky se používají např. frame layout, linear layout, table layout nebo relative layout. **Frame layout** je nejjednodušším zobrazením, neboť bez nastavení atributu layout_gravity se jednotlivé prvky pouze skládají do levého horního rohu přes sebe. **Linear layout** (horizontální a vertikální) rozdělí obrazovku na jednoduché pruhy, ať už svislé nebo vodorovné, a jednotlivé prvky se řadí postupně jeden za druhým. Takové rozložení je velmi vhodné např. pro seznam mož-

ností nějakého menu. **Relativní layout** se chová vice jako webová stránka s pozicováním dle css vlastnosti position-relative. Určuje se, kde by se měl prvek objevovat vzhledem k obrazovce. Table layout strukturuje prvky do tabulky. Důležitou vlastností pak je možnost vkládání jednotlivých layoutů jeden do druhého, čímž se přesněji vyspecifikuje konkrétní chtěné rozložení (obrázek 4). Jak lze vypozorovat, pracuje se zde s **LayoutParams**. To je velmi důležitá vnořená třída ViewGroup (tedy rozšiřující ViewGroup.LayoutParams), jež umožňuje jakékoliv nastavení parametrů daného layoutu.



Obrázek 5: Příklad skládání layoutů (8)

Z ukázky výše je vidět, že do layoutu lze vložit nejen triviální prvky typu jednoduchý text, ale Android podporuje všemožné formulářové prvky, různé lišty pro zobrazení průběhu nějaké akce (progressbar), číselníky a widgety. Dále lze hovořit o speciálních view např. pro ukázání mapy, nebo view s posouváním mezi obrazovkami doleva, doprava a mnoho dalších.

1.4 Další typy komponent

Stejně jako je rozmanitost v prvcích zobrazení, i komponenty nemusí být vždy nutně jen Aktivita zobrazující nějaké view. Občas je třeba spustit nějakou akci na pozadí, získat nějaká data bez jejich zobrazení atp. Je tedy vhodné se o několika takových typech komponent zmínit.

Služba dle Android Developeru (4) se vyznačuje tím, že dlouhodobě běží na pozadí a provádí nějakou činnost. Dá se tedy říct, že je to Aktivita bez rozhraní. Mezi typickými scénáři je například přehrávání hudby v době, kdy je uživatel v jiné aplikaci, nebo stahování dat na pozadí. Je třeba ale rozlišit službu pro stahování dat a samotnou správu dat.

Content provider spravuje sdílená data. V zásadě umožňuje pracovat s některými prostředky, které aplikace využívá. Data se pak vytahují a mění v závislosti na právech k daným datům. Mezi nimi si lze představit např. data uložená do souborového systému, databáze skrze SQLite, ale i jiné persistentní (trvalé) úložiště. Jako příklad se udává třeba využití informací o telefonních kontaktech. Ty jsou dostupné z více aplikací a díky tomu je možné je využívat a doplňovat viz např. populární WhatsApp[10].

Broadcast Reciever přijímá a reaguje na systémové a jim podobné zprávy. Např. že se stáhnul obrázek, že baterie mobilu je na konkrétní nízké hladině a jiné informace. Ač broadcast reciever nemá vlastní view, na základě informací dokáže vytvořit notifikaci, jež se pak zobrazuje v notifikační liště jako zpráva.

1.5 Struktura

Než bude probráno konkrétní nastavení, je vhodné se blíže podívat na souborový systém aplikace. Android používá vysoce organizované a úsporné rozdělení pro jakékoliv prvky použité uvnitř sebe. Nastavení programu není výjimkou, provádí se na více místech v závislosti na jeho charakteru a z důvodu úspory a univerzálnosti využívá formátu xml. Je třeba upozornit, že struktura souborového systému má více variant v závislosti na API[11] a podporované verzi operačního systému Android. Základní filozofie ale zůstává napříč variantami stejná.

Zobrazená struktura navíc může být detailnější. S přihlédnutím na komplexnost programu podporuje Android nejen samostatné složky pro různá zařízení, nýbrž i namespace[12] a rozdělení na více částí (frontend, backend), moduly atd.

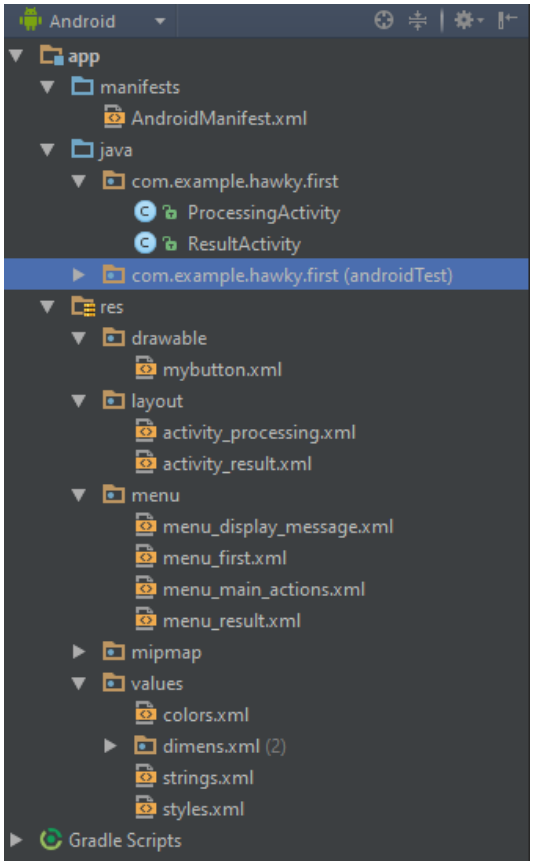
---

[10] Aplikace pro posílání zpráv a obecnou komunikaci skrze telefonní číslo.

[11] Zkratka pro vývojové prostředí jako například Android Studio, Eclipse nebo NetBeans

[12] Unikatní systém identifikátorů- v jednom balíku či **jmenném prostoru** se nesmí opakovat stejné id.

Obrázek 6 ukazuje strukturu, se kterou pracuje aplikace v praktické části a která je sdružena do adresáře app.



Obrázek 6: Anatomie Android aplikace

App se rozpadá na prvky názorně demonstrující jednotlivé funkce aplikace a popsané detailněji například v Software And Technology Education (9):

- **Manifest** – nejdůležitější soubor aplikace, deklaruje podmínky použití, slouží pro základní nastavení (viz kapitola 1.6 Nastavení)
- **Java** – jak název napovídá pro Java třídy a tedy jádro aplikace - Aktivity
- **Res** – neboli **resources** (zdroje), zde se ukládají pomocné soubory aplikace - především XML soubory, ale lze zde mít i obrázky a další. Podle charakteru se dále dělí na:

- **Drawable** – Zde se ukládají bitmapové obrázky popř. XML soubory, které reprezentují grafické prvky aplikace[13]. Zajímavostí této složky je, že umožňuje podsložky drawable-hdpi, drawable-ldpi atd. dle nastavení skupin zařízení z kapitoly 1.3 Zobrazení. Tím se technicky řeší obrázkové prvky pro různé typy obrazovek.
- **Layout** – nejdůležitější část zdrojů, ukládají se zde **View** aplikace.
- **Menu** – XML soubory definující menu včetně kontextových, menu voleb, sub menu a dalších.
- **Mipmap** – slouží k uložení obrázku ikony spouštěče aplikace. Opět lze jako v drawable nahrát pro různá zařízení různé ikony[14].
- **Values** – pomáhá udržovat aplikaci přehlednější tím, že se zde dle charakteru informací ukládají do XML **hodnoty** použité v aplikaci jako názvy a obecně textové řetězce (strings.xml), styly a jejich definice (styles.xml), barvy (colors.xml) a nastavení velikostí (dimens.xml).

1.6 Nastavení

Výše zmíněná anatomie napovídá, že velká část struktury aplikace je věnována nastavení. To je představováno základním nastavením pomocí XML souboru **manifest**, a dílčími nastaveními pomocí **resources** dle jejich povahy.

Jak již bylo řečeno, manifest je nejdůležitějším souborem aplikace. Shromažďuje v sobě klíčové informace, kterými se řídí spuštěná aplikace a i samotné zařízení s daným programem pracuje dle uvedených informací. V manifestu se deklarují především:

- Informace o **kompatibilitě**. Při startu projektu je třeba se rozhodnout, jaké verze Androidu budou podporovány a podle toho přizpůsobit strukturu a funkce programu. Taktéž lze definovat, pro jaký typ zařízení je aplikace určena viz např. skupiny zařízení z kapitoly 1.3 Zobrazení

[13] v tomto případě např. reprezentuje tlačítko, které se **vykresluje** (drawable = možno vykreslit) při startu aplikace

[14] Rozdělení se ale neřeší ukládáním do různých složek, ale ikona je uložena ve 4 různých souborech.

- Informace o **prostředcích (uses permission)**, které program používá a ke kterým potřebuje svolení uživatele. Mimo jiné se jedná o seznam, jenž se objevuje například v Obchodu Play[15] před stahováním aplikace, ale stejně tak při instalaci/prvním spuštění programu. Jako typický příklad lze uvést povolení, že aplikace smí využívat ke svému chodu internetové připojení, právo na čtení/zápis na externí úložiště (např. paměťová karta telefonu), nebo jiné hardwarové a softwarové funkce zařízení (jako senzory, kameru, bluetooth, mapy, SMS atd.).
- Základní **parametry** aplikace – název, cesta a jméno ikony, styl (theme), název balíku (package) pro určení cesty v souborovém systému ad.
- Seznam **Aktivit** s nastavením main activity, aby aplikace poznala, odkud začít - tedy co je spouštěcím souborem programu. U všech aktivit pak uvádí případné další individuální parametry.

Ve zdrojích lze nalézt méně podstatná nastavení doplňkového charakteru tzv. values. Ty mají úlohu nejen sběrnou, ale i nastavující. Např. Styles.xml funguje nejen jako seznam použitých stylů, ale zároveň jako stylopis vzdáleně podobný CSS u webů. Doporučuje se, obzvláště v případě vícenásobného použití, aby se konkrétní prvek stylu (např. velikost písma) udržoval v tomto souboru centralizovaně. Podobně jako styles funguje i soubor dimens.xml, který navíc lze udržovat ve více variantách zároveň, s ohledem na skupinu zařízení.

Nejvíce upravovaným souborem values jsou pravděpodobně strings.xml. Zde se ukládá seznam textových řetězců, což zpřehledňuje views a activity. Navíc často používané položky jako názvy tlačítek, seznamových položek atp. se lépe udržují a upravují z centralizované struktury. Proto je doporučeno, než používat jednotlivé řetězce přímo v kódu, držet všechny ve strings souboru. Podobně pak fungují i colors.

Aby systém zástupných entit dobře fungoval, je třeba správně využívat další podstatnou funkci Androidu – **ID**.

[15] Online manažer pro Android aplikace, hlavní zdroj nových programů pro uživatele, obdoba iStore pro Android od firmy Google a tedy využívající e-mailový účet.

Ty se používají nejen pro seznamy výše a jsou globální. Všechny položky z values mají jednoznačné identifikátory, které pro volání v Aktivitách, manifestu, nebo view nahrazují konkrétní hodnoty. Volání se pak liší dle užití:

```
android:text="@string/button_start" /* příklad volání uvnitř view a manifestu */
setMessage(getString(R.string.downloading)); /* příklad volání uvnitř Aktivity */
setContentView(R.layout.activity_processing); /*volání view v Aktivitě */
```

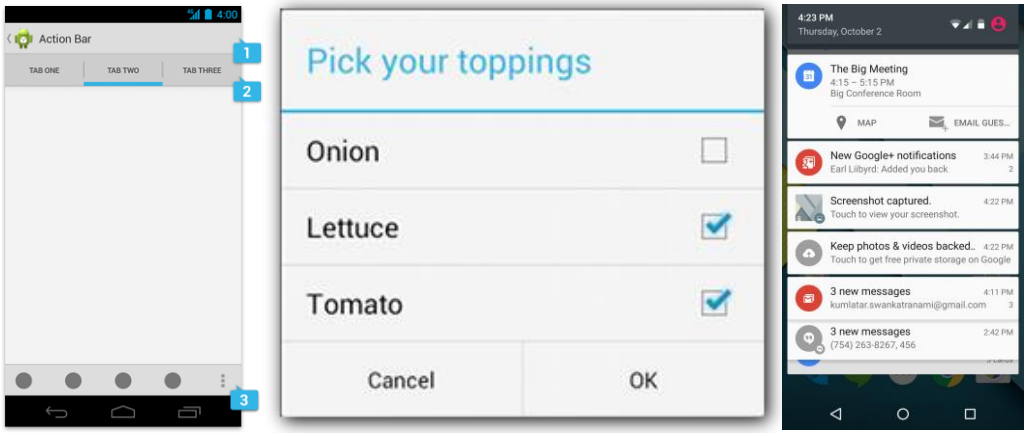
Ukázka 2 – Volání dle ID v závislosti na umístění

R. znak určuje, že prvek je typu android.R.attr, tedy že prvek je referencí na objekt aplikace podobně jako je využito v ukázce 2.

Pro volání menu uvnitř Aktivity je třeba mít implementovanou metodu onOptionsItemSelected(MenuItem item), která taktéž využívá ID systém. Její položky jsou vedené v resources v podsložce menu.

1.7 Uživatelské rozhraní

Jak bylo zmíněno výše, v Androidu je možné vytvářet a upravovat **menu** různých podob. Od klasické horizontální/vertikální nabídky, přes sub menu, kontextové nabídky v rámci možností v dané situaci, až po vyskakovací menu. Speciálním případem menu od verze Android 3.0 je **action bar**, který v zásadě funguje jako menu při horním a/nebo spodním okraji, většinou ve tvaru lišty neboli horizontální nabídky akcí.



Obrázek 7: Příklady uživ. rozhraní (zleva action bar, dialog a status notification) zdroj (4)

Dalším oblíbeným prvkem jsou **dialogy**, které se zjevují v případě potřeby nějaké volby nebo pro předání informace stavu aplikace. Z nejpoužívanějších možností lze zmínit nabídky typu OK/Cancel (**alert dialog**) nebo zobrazení průběhu stahování souboru tzv. **progress dialog**. V dialozích se často objevují formulářové prvky podobné webovým. V neposlední řadě lze ke komunikaci s uživatelem zvolit **status notification**.

Notifikace se objevují jako zprávy pod horním okrajem displeje v aplikačním rozhraní mimo konkrétní aplikaci v tzv. notifikační oblasti. Tyto zprávy mohou být vygenerovány např. pomocí broadcast receiveru pro upozornění na nějakou událost.

1.8 Komunikace s okolím

Stejně jako má Android různé možnosti ovládání, i připojení je pro něj zcela zásadní a unikátní záležitost. Není totiž běžné, aby nějaký přístroj měl takové možnosti komunikace se světem, jako mobilní zařízení. Samozřejmě hraje svoji roli daný typ a model hardwaru, ale v zásadě se dá propojení se světem rozdělit na připojení pomocí:

- **GSM** - mobilní telefonní síť - k těm se vztahují mobilní internetová připojení zajišťována operátorem GSM viz podkapitola 2.1 Hardware.
- **Wi-Fi** - internetová bezdrátová síť.
- **Bluetooth** - komunikace na krátkou vzdálenost, pro přenos dat mezi zařízeními ve vzdálenostech do 100m (Bluetooth class 1), běžné použití v řádu metrů, lze použít i pro internet.
- **Infračervený port a NFC**[16] - komunikace na krátkou vzdálenost, pro přenos dat mezi zařízeními v centimetrových vzdálenostech (nevhodné pro internetové připojení).
- **GPS** - zjišťování polohy zařízení (nevhodné pro internetové připojení).
- **USB** - skrze rozhraní usb je možné přímé propojení s dalším zařízením a výměna dat. Pomocí něj lze sdílet internetové připojení.

Pro využití výše zmíněného hardware, má Andriod implementované různé metody. Pro zjištění stavu sítě se volá třída **ConnectivityManager**, konkrétněji pak:

[16] **Near field communication** - rádiová bezdrátvá komunikace do 4 cm od zařízení

```
(ConnectivityManager)getSystemService(Context.CONNECTIVITY_SERVICE);
```

Poté se smí volat metoda getNetworkInfo(), která dokáže prozradit nejen to, zda je zařízení připojeno, ale také pomocí kterého hardwaru. Trochu matoucí ale může být, že např. na stav `connManager.getNetworkInfo(ConnectivityManager.TYPE_WIFI);` smí metoda navrátit kladnou odpověď, i když nemá aplikace přístup na webové stránky. Kladná odpověď totiž říká, že zařízení je připojené k Wi-Fi síti, tedy k Wi-Fi routeru, což ovšem nevypovídá o tom, zda samotný router je připojený k internetu, pouze, že jste součástí vnitřní sítě, byť je to pravděpodobný předpoklad.

Manipulace s daty z internetu lze pak provést skrze **HttpURLConnection** a abstraktní třídy InputStream/OutputStream. Jednoznačně je vhodné zmínit, že taková činnost by se rozhodně neměla dít v hlavním vlákně aplikace, tedy v běžící Aktivitě. Důvodem je fakt, že by to mohlo brzdit chod samotné aplikace, zbytečně navyšovat nároky na paměť, popř. způsobit pád takového programu. Proto se k takovému účelu používá **AsyncTask**. Ten otevře nové vlákno aplikace a na pozadí provede danou akci. To je stabilnějším řešením jak pro zpracování případných chybných stavů, tak šetrnějším k hardwarovým prostředkům zařízení. Výhodou AsyncTasku je i nativní podpora 4 fází úkolu. Fáze před spuštěním - pro přípravu hodnot, samotné spuštění úlohy, onProgressUpdate() pro aktualizaci stavu a samozřejmě poslední ukončovací fázi, která zajistí předání případných informací a spuštění dalších metod. AsyncTask se nicméně hodí spíše na menší úkoly. Při využití na více než několik sekund Android developer (4) doporučuje spíše balíky jako Executor, ThreadPoolExecutor nebo FutureTask.

1.9 Upload

Při odesílání souboru se situace malinko ztěžuje o to, že data jsou přijímaná mimo prostředí Android, čímž vyvstává potřeba stanovit komunikaci mezi zařízeními. Typ komunikace závisí na typu serveru. Existují speciální řešení na umístění jako je Dropbox, FTP server, Facebook atd. V případě referenčního programu této práce bylo zapotřebí nastavit komunikaci skrze webový Apache PHP server, přičemž Apache nabízí zdarma balík funkcí pod jménem HttpComponents (10).

Je třeba si dát zvýšený pozor, aby byl daný server nastavený tak, aby přijímal komunikaci a reagoval na podmět Android aplikace. Za situace, že je třeba nějaká komplexnější informace od serveru, než je samotné přijetí souboru, lze na straně serveru v daném programovacím jazyce vytvořit soubor, který taková data poskytne. To ale znamená přidat do Androidu aplikace další metody pro očekávané odpovědi pomocí HttpResponse a jiných.

Standardně je zapotřebí využívat komunikačních prostředků, kterým daný server bude rozumět. To buď zajistí výše zmíněný podporovaný balík, nebo se v případě webového serveru využívá klasických metod (setRequestMethod) pro přijetí souboru GET a POST, známých například z formulářového prostředí. Takto lze po vyplnění hlavičky informacemi v požadované formě dostat soubor na server.

Těmito popsanými pravidly a know how se bude řídit i program přidružený k této diplomové práci. Dříve, než bude tento projekt konkrétněji vysvětlen, je zapotřebí teoreticky představit další tématický okruh - měření rychlosti obecně.

2. Měření připojení

Aplikace, jež byla vyvíjena v propojení na tuto diplomovou práci, je zaměřena na měření rychlosti internetového připojení, nicméně jak takové měření provést a jak získat relevantní výsledky zatím nebylo zmíněno. Je také dobré si osvětlit, o jakých rychlostech je v reálném světě řeč.

2.1 Hardware

Mobilní přístroje nynějška (červen 2015) operují v případě nejvyšší řady s 8 jádrovými procesory (2,1GHz, 14nm), 3GB RAM a verzí Android 5.0 (Lollipop). Takovými parametry oplývá ale jen zlomek mobilních telefonů a tabletů. Reálné hodnoty běžných zařízení se pohybují okolo 2-4 jádrových procesorů mezi 1-1,5GHz, 1-2GB RAMM, navíc s nevelkým prostorem pro uložení. Je zde řeč o kapacitě kolem 16GB – 32GB a verzí Androidu 4.X. Pro měření rychlosti internetu jsou pak klíčové hodnoty internetového stahování a posílaní dat, kde se hodnoty liší podle typu připojení.

Při připojení pomocí Wi-Fi je v zásadě nejvíce omezujícím faktorem rychlost samotné sítě a síla signálu. Bezdrátová síť teoreticky nabízí 1800 Mb/s (dle standardu

IEEE 802,5.11ac, který ale opět podporuje zatím jen několik přístrojů). Častějšími čísly jsou řádově desítky a jednotky Mb/s už v důsledku nabídky operátorů internetových připojení.

Název technologie	Rok uvedení	Přenosová frekvence	Maximální teoretická rychlost dat. Přenosu	Způsob přenosu dat	Max. vzdál. vysílače a přijímače
GPRS (síť GSM)	1997	900/1800 MHz	80 kbps	Symetrický / Asymetrický	35 km
EDGE (síť GSM)	2004	900/1800 MHz	218/134 kbps	Symetrický / Asymetrický	30 km
1xRTT (síť CDMA2000)	2004	450-2100 MHz	307/153 kbps	Asymetrický	54 km
1xEV-DO (síť CDMA2000)	2004	450-2100 MHz	3.1/1.8 Mbps	Asymetrický	54 km
UMTS (W-CDMA) (síť UMTS)	2000	1885-2200 MHz	2048 kbps	Symetrický / Asymetrický	2 km
HSDPA (HSDPA+) (síť UMTS)	2004	873/1900 MHz	14.4 (42 Mbps)	Technologie pouze pro Downlink	6 km
HSUPA (HSUPA+) (síť UMTS)	2005	873/1900 MHz	5.76 Mbps (7.2 Mbps)	Technologie pouze pro Uplink	5 km
LTE (síť E-UTRAN)	2008	V Evropě obvykle 800, 1800, 2600 MHz	100/50 Mbps	Symetrický / Asymetrický	30 km

Obrázek 8: Tabulka rychlostí bezdrátových mobilních připojení (11)

Při použití GSM sítě se pak rychlosti pohybují dle obrázku 8. Teoreticky je u nových zařízení na některých místech v republice k dispozici LTE. Ta dosahuje reálných hodnot 13Mb/s download a 10Mb/s upload (12), což je v tuto chvíli praktický strop, který je možno u GSM změřit.

Tímto je určena horní hranice. Spodní hranicí se dá předpokládat rychlost blízká nebo rovná nule. To se stává především při velmi špatném signálu, jenž neustále padá nebo nezvládá navazovat spojení.

2.2 Problematika měření

Z výše uvedených dat vyplývají některé problematiky, které ztěžují samotné nastavení logiky práce aplikace. Navíc další přibývají při postupném návrhu řešení. Proto se bude při vymýšlení návrhu postupovat od nejjednodušší možné varianty. Pomocí ošetření různých faktorů se bude návrh optimalizovat až k výsledku s dostatečnou relevancí měření, viz následující úvaha a porovnání.

Nechť existuje nejprimitivnější způsob měření rychlosti internetu. Toho by bylo docíleno tak, že se z internetu stáhne soubor, změří se jeho velikost a doba, po kterou byl stahován. Protože rychlost internetu je prezentována skrze download i upload, budiž i druhá část aplikace, která pak soubor nahraje na externí místo na internetu a opět nechť je změřena doba, která k tomu byla potřeba. Relevantnost výsledku je již na první pohled ohrožena (nejen) těmito faktory:

- Soubor byl příliš malý, než aby se projevila plná rychlost sítě.
- Soubor byl stahován v době, kdy byl výkyv sítě, tedy výsledek sice může být přesný, ale jen v dané chvíli. Tím se zkreslí představa průměrného maximálního možného výkonu v dané oblasti, protože uživatel nerozlišuje čísla výkyvu a standardního signálu.

Proto se nabízí způsob číslo 2. Stahovat natolik velký soubor, jehož velikost by byla dostatečně velká a tedy průkazná i při variantě nejrychlejších typů připojení zmíněných výše. Takový přístup nebude tolik náchylný k výkyvům sítě. Hned ovšem přichází na mysl otázka, co když zrovna půjde o pomalý typ připojení a příliš dlouhé stahování bude komplikovat dokončení samotného měření.

Dále se nabízí varování ohledně uživatelské přívětivosti. Na kolik je relevance důležitější než doba spuštění aplikace. Příliš dlouhé spuštění nebude uživatel tolerovat a program se tak lehce stane nepotřebným a nepraktickým.

Budiž tedy případ číslo 3. Jak bylo řečeno v podkapitole 1.8 Komunikace s okolím, lze zjistit, jakým způsobem je dané zařízení připojeno a na základě obrázku 8 výše, lze odvodit, jak velký soubor je potřeba k dosažení výsledku. Tím se eliminuje příliš dlouhé stahování. Tato metoda je ovšem stále náchylná na časovou prodlevu při navázání a ukončení spojení. Časová doba je navíc jen odhadnutá. V případě nepříliš silného signálu jinak rychlé sítě stále není plně řešena potíž s dlouhým stahováním.

Proto se rozšíří myšlenka na případ 4. Nechat stahovat obrovský soubor, ale ukončit stahování po určité době, pro ilustraci lze zvolit 10 nebo 20 sekund. To zajistí dostatečné měření nezávisle na typu připojení.

U měření rychlosti nejde o přenos samotného souboru, ale spíše o rychlost pohybu dat. Z principu věci je tedy měřenou veličinou tok dat a v tomto případě je měřena

v jednotkách Mb/s. To vede k možnosti využití bufferu, u něhož lze měřit, jak rychle se naplňuje, což znamená, jak rychle se data posouvají určitým směrem. Při dostatečně dlouhém a detailním měření, by se takto dal vysledovat trend stahování, tedy jaká rychlost byla na počátku stahování a v průběhu specifického času. Výhodou je také přesun jen takového množství dat, které síť zvládne. Vzhledem k možnostem sítí, nebude pravděpodobně tato hodnota tak vysoká, jako v případě stahování celého souboru, byť třeba i řádově menšího.

Nezávisle na předchozí myšlence lze vytvořit i případ 5. Nejít cestou stahování jednoho velkého souboru, ale spíše více malých v časových rozestupech. To má výhody v menším objemu dat a menším zatížení zařízení. Je několik metod, které se dají u takového přístupu aplikovat. Lze posílat stejně velké soubory, nebo lze jejich velikost postupně navyšovat (10kB, 100kB, 1MB). Opět aby byly reflektovány aktuální podmínky, je možné velikost dalšího testovacího souboru upravit na základě výsledku předchozího, čímž se optimalizuje časová stanovená doba stahování/posílání. Technicky vzato se dá aplikovat i časový limit jako v předchozím případě, jen se hůř dodržuje v případě nestabilní sítě.

Nevýhodou celé této myšlenky se jeví fakt, jenž je zároveň i otázkou, co se vlastně míní rychlostí připojení. Čím více souborů se totiž použije k měření, tím víc se do samotné rychlosti započítává i čas navázání a ukončení spojení se serverem. Při rychlém internetu se dle praxe pohybuje do rychlosti v řádu desetin sekund[17]. Při výpočtu tedy s 1Mb/s a 0,1ms je rozdíl za 10 sekund měření 10Mb a 9,9Mb, při pěti malých souborech se pak rychlost teoreticky upraví až na 9,5Mb, což už je relativně výraznější rozdíl oproti původnímu výsledku (5%). Při samotném využívání internetu sice k tomuto efektu dochází také, tedy původní myšlenka nabádá k započtení takové doby do celkové. Na druhou stranu, při hlubším zamyšlení, je to, co se měří, maximální rychlost, tedy co je strop možností dané sítě a maximum, co dokáže síť v daném čase.

Navíc o to těžší situace nastává, pakliže síla signálu není na vysoké úrovni. Potíž navázání/ukončení spojení je v tzv. 3-way handshake[18] dle InetDaemon.Com (13). Ten

[17] Přesné číslo je špatně měřitelné při tak nízkých hodnotách a nebylo potřeba zjišťovat přesněji.

[18] V případě ukončení je to 4-way handshake.

je potřebný v obou případech a v zásadě jde o tři/čtyři fáze, jedna navazující na předchozí, kde si server a zařízení vyměňují packety pro zjištění stavu a dohledání jeden druhého. Packet je třeba nejdříve odeslat, pak přijmout a následně opět odeslat. Všechny packety musí najít cestu a na rozdíl od samotného stahování, jednotlivé packety musí dorazit v daném pořadí a nejsou odeslány, dokud není přijmuta předchozí část. O to více se tento faktor projeví při odesílání souboru, neboť tam se čeká navíc na odpověď serveru, zda soubory dorazily v pořádku. V případě měření na straně serveru se navíc potřebují předat hodnoty z měření rychlosti, což znamená extra připojení navíc. Je záhodno totiž měřit rychlost na té straně, kam data putují, protože to přesněji vypovídá o rychlosti toku dat.

To vše může při nestabilním připojení působit výraznější časové ztráty, což se může projevit na výsledku měření. Na obranu této měřící metody je dobré říct, že do jisté míry jsou všechny metody zatížené chybami z výkyvů sítě. I při dlouhém měření dle řešení 4, kdy se půl doby využije internet na maximum a půl špatně v důsledku nějaké anomálie, pak hodnota měření bude skoro poloviční oproti opravdové rychlosti sítě. Na druhou stranu z eticko logických důvodů, není asi důležitým cílem dávat uživateli informaci o teoretickém maximu sítě, ale spíše o schopném reálném využití, jakého je síť v dané chvíli schopna. V tomto ohledu jsou obě řešení na dostatečně uspokojivé úrovni, aby se daly použít. Přístup s více soubory navíc skýtá větší statistické možnosti a tím nabídne zase jiný pohled na data v závislosti na dílčích výsledcích. Takové statistiky jsou ovšem možné i v případě jednoho souboru, pakliže se měření rozdělí na více dílů např. podle výsledku načítání bufferu, čímž lze taktéž velmi efektivně sledovat trend přesunu dat. Je možno konstatovat, že metody 4 a 5 jsou dostatečně zajímavé pro porovnání a případné zpracování.

2.3 Výběr metody

První metody logicky není třeba srovnávat, protože byly vyřazeny již v procesu zamýšlení se nad problematikou. Ke zpracování připadají v úvahu pouze poslední zmíněné možnosti, přičemž slovně již byly porovnány. Prioritou pro výběr je jednoznačně přesnost (proto zvýraznění), poté uživatelská přívětivost. Do tabulky byly také zavedeny dílčí vlastnosti ovlivňující přesnost, aby bylo zřejmé, jakým způsobem byla vyhodnocena. Je zde vidět snaha zavést všechna výše zmíněná kritéria ovlivňující celkový

výsledek. Vzhledem k tomu, že přesnost v závislosti na uživatelské přívětivosti nejlépe splňuje metoda 4, bude použita i pro samotnou aplikaci.

Zde je samotné porovnání:

Vlastnost \ Metoda	Metoda 4	Metoda 5
Přesnost	+	-
Náročnost na paměť	-	+
Náročnost na výkon	-	+
Náročnost na čas	+	+
Uživatelská přívětivost	+	+
Náchylnost na výkyvy	+	+
Citlivost na rychlost sítě	+	+
Citlivost na 3-way handshake	+	-
Statistické možnosti	+	+

Tabulka 3: Porovnání metod měření

3. Praktická část

Pro pochopení programovacích postupů v jakémkoliv prostředí je nejdůležitější praktická zkušenost. Proto po představení teoretických znalostí přichází část, kde bude možnost zužitkovat nastíněné informace. Ta bude doložena jak empirickými znalostmi, získanými vytvářením referenčního projektu, tak příklady kódu a jejich okomentování k přiblížení jednotlivých částí programu a jejich významu.

3.1 Zadání

Téma práce je aplikace pro měření rychlosti připojení k internetu se zadáním:

- Programovací jazyk Java pro Android v prostředí Android Studio.
- Podpora Androidu od 4.X
- Podpora bezdrátového Wi-Fi i mobilního GSM připojení.
- Podpora zjištění stavu připojení před samotným měřením.
- Podpora rozdílných typů displejů.
- Podpora měření rychlosti dle metody 4 z podk. 2.2 Problematika měření

- Podpora měření stahování i odesílání.
- Rozvržení aplikace bude dle návrhu z podkapitoly 3.4 Rozvržení.
- Aplikace bude podporovat dialogy pro informaci o stavu měření.
- Aplikace bude v tmavém provedení v AppCompat stylu.

3.2 Dokumentace

Po zprovoznění prostředí je nejvhodnější nativní informační kanál Android Developer (4). Nejen že obsahuje základní informace pro zprovoznění API, ale je zde detailní popis všech prvků Android prostředí a velice cenný tutoriál, jenž ukázkově provádí světem vývoje mobilních aplikací od první Aktivity po sofistikovanější projekty. Velmi dobrým pomocníkem se při řešení konkrétních situací nabízí, při správném zadání klíčových slov, server http://stackoverflow.com/, kde je mnoho použitých situací vcelku zajímavě rozebráno a i leckdy rozumně odůvodněno. Dále lze vycházet např. ze zdrojů zmíněných v Seznam literatury. Samotnou pomoc a možnosti nabízí i Android Studio a jeho nápověda v průběhu programování.

3.3 Prvotní nastavení

Po instalaci zřejmých prvků[19] nastává čas pro samotný projekt. Ten se spustí skrze nabídku a po vybrání jména, API 14 Android 4.0 (IceCreamSandwich), Blank Activity a vybrání jména hlavní Aktivity (v tomto případě ProcessingActivity) je projekt připraven. Dobrou vlastností Android Studia je, že vytvoří rovnou prvotní funkční program včetně view tzv. Hello world!. Na něm lze otestovat, zda je správně zprovozněná testovací zařízení jako Android mobil/ tablet popř. emulátor[20].

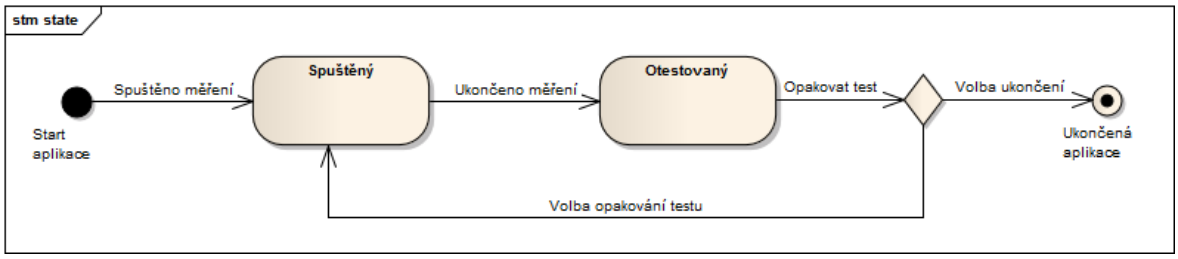
3.4 Rozvržení

Obecné pravidlo, zmíněné v 1.2 Základní struktura, říká, že na každou obrazovku je třeba nová aktivita a view. Pro účely referenčního projektu bude potřeba minimálně uvítací obrazovka, která představí účel aplikace, bude zde spouštěcí tlačítko pro zahájení měření. Další obrazovka budou progress bary pro informování o stavu

[19] Instalace Javy JDK a Android Studia, jeho zprovoznění a nastavení testovacích zařízení.

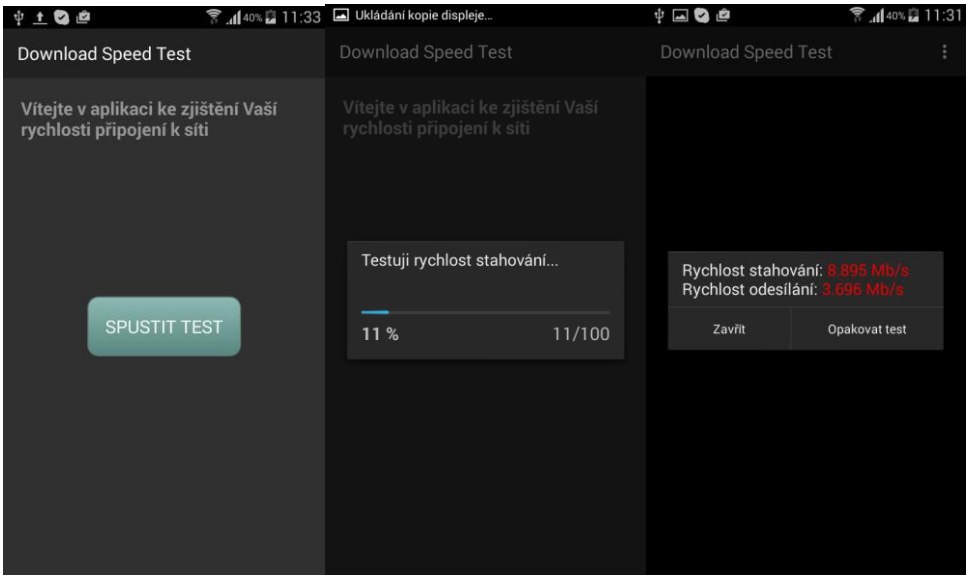
[20] Emulátor je součástí Android Studia, ale existují i externí varianty. Je to simulátor mobilního zařízení, kde lze vybrat libovolný Androidí typ zařízení a na obrazovce pc skrze myš testovat chování app.

měření a v poslední obrazovce bude dialog s výsledkem měření. Celý průběh programu lze znázornit skrze stavový diagram:



Obrázek 9: Stavový diagram aplikace

Při prvním vytváření aplikace existovaly tedy 3 Aktivity a pohledy. Protože každá aktivita musí obsahovat základní povinné metody onCreate, onCreateOptionMenu a onOptionsItemSelected[21], což zabralo většinu kódu první Aktivity a skoro ten samý obsah Aktivity se musel vygenerovat i v další Aktivitě (jen vyjma ovládání tlačítka), není asi potřeba exaktně dělat novou Aktivitu. Taktéž progress bar potřebuje nějaké view, nad kterým se ukáže[22]. Ten by byl redundantním s již existujícím view, tedy ani nové view nebylo potřeba a aplikace se tak zkrátila na dvě Aktivity, i když se opticky zdá, že se používají 3 pohledy.



Obrázek 10: Ukázky obrazovek aplikace

[21] Metody, které se v Android Studiu skrze volbu New Activity vytváří automaticky.

[22] Ze své podstaty se obecně dialogová okna zobrazují nad částečně překrytým obsahem view

3.5 Manifest

Dalším krokem je nastavení základních parametrů. Android Studio díky postupu z 3.3 Prvotní nastavení, již vyplnil základní strukturu s parametry. Celý soubor je ve formátu XML a má přesně danou formu z důvodu kompatibility různých verzí Androidu. Tato informace je pak i uvedená skrze řádek uses-sdk:

```
<uses-sdk android:minSdkVersion="4" android:targetSdkVersion="14" />
```

V dalším řádku je možnost přidání povolení na různé prostředky zařízení. Android ve snaze o větší transparentnost a funkční udržení jádra defaultně zakonzervovává aplikaci, aby neměla možnost využívat citlivější funkce jako přístup internetu atd. Protože z povahy aplikace nebývá vždy jasné, jaké prostředky bude potřebovat, při instalaci programu Android zkontroluje, jaká práva jsou uvedena v Manifestu. S těmi poté systém seznámí uživatele, který musí dát výslovný souhlas s použitím takových funkcí. Při programování navíc Android Studio hlídá, zda se nevyužívají balíky a metody, které vyžadují taková oprávnění. V případě, že ano, sám je nabídne programátorovi k přidání, čímž je sám dovyplní právě zde do Manifestu. Syntaxe a seznam všech oprávnění je k dispozici v (4). Pro potřeby referenčního webu to jsou:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"…
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

Ukázka 3 – Uses-permission v Manifestu

Nastavené proměnné povolují komunikaci s internetem, se zjišťováním Wi-Fi, síťového a mobilního stavu a práci s úložištěm v mobilu pro posílání a ukládání dat.

Dále je nutné vyplnit informace o detailech aplikace a všech Aktivitách programu, neboť jinak je systém nebude využívat. Spouštěcí Aktivita je již vyplněna a v zásadě ji není třeba měnit. Postupem času pak zde budou přibývat další, nicméně ty opět dokáže vyplňovat Android Studio samo, pakliže programátor využije přidání Aktivity skrze nabídku New Aktivity v menu. Nastavení pro jednotlivé aktivity se dá rozšířit o

další možnosti např. podle rozdělení na jednotlivé balíky atd. Pro účely referenčního projektu to stačí takto.

```
<application
   android:allowBackup="true"
   android:icon="@mipmap/ic_launcher"   // Nastavení ikony aplikace
   android:label="@string/app_name"     // Nastavení názvu aplikace
   android:theme="@style/AppTheme" > // Nastavení stylu aplikace
 /* Spouštěcí aktivita */
  <activity
      android:name=".ProcessingActivity"
      android:label="@string/app_name" >
      <intent-filter>
         <action android:name="android.intent.action.MAIN" />
         <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
   </activity>
/*Dodatečně přidaná aktivita*/
   <activity android:name=".ResultActivity"></activity>
</application>
```

Ukázka 4 – Nastavení aplikace a Aktivit v Manifestu

3.6 První obrazovka

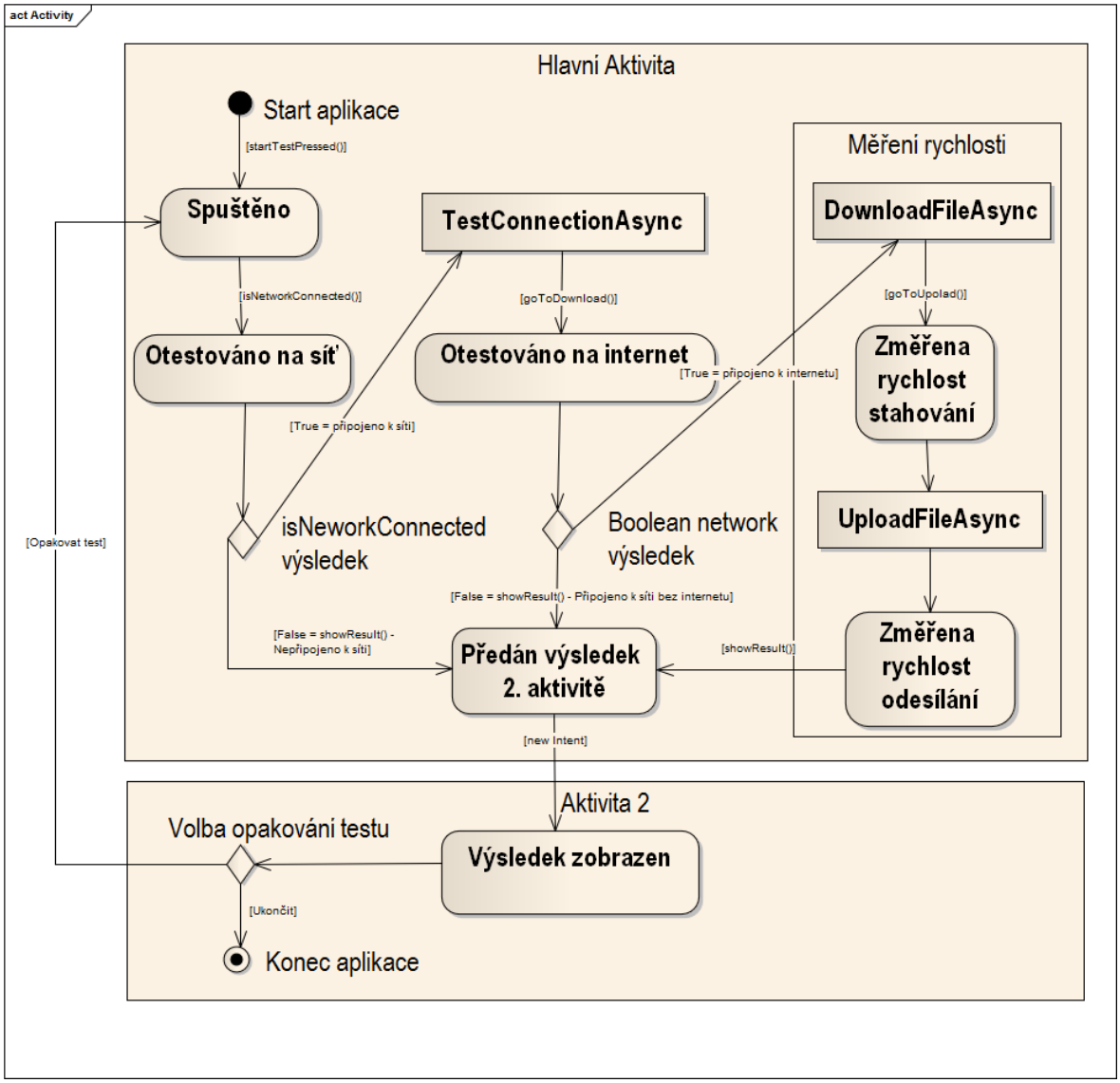
Po nastavení manifestu nastává chvíle pro samotné programování chování. Protože byla kvůli větší přesnosti zvolena složitější varianta testu, pro zpřehlednění je příhodné ukázat, co je cílem snažení a jak se celá aplikace bude chovat.

Aplikace potřebuje změřit stahování a odesílání. Aby to mohla provést, potřebuje ověřit, že je v síti, která je připojena k internetu. Postup tedy je:

- ověřit, zda je připojeno k síti
- ověřit, zda je připojeno k internetu
- změřit rychlost stahování
- změřit rychlost odesílání
- podat zprávu o výsledku a nabídnout nové měření

Klíčové jsou zde 3 AsyncTasky, jež se postupně připojují k internetu a na samostatných vláknech provádí činnosti dle postupu.

V případě, že zařízení nemá k dispozici síť, nespustí proces ani jeden z AsyncTasků, jak znázorňuje diagram aktivit.



Obrázek 11: Podrobný diagram aktivity aplikace

První obrazovka má příhodně podobný a jednoduchý charakter jako kapitola Building Your First App, kterým Android Developer začíná svůj trénink pro začátečníky. V prvním pohledu bude jednoduchý úvodní text, tlačítko pro spuštění aplikace a taktéž by bylo dobré, aby byl v horní části obrazovky vidět název spuštěného programu. Z pohledu rozložení obrazovky, text by měl být nahoře, dostatečně viditelný a tlačítko uprostřed (nehledě na zařízení).

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:orientation="vertical"                    //Skládá prvky pod sebe
   android:layout_width="wrap_content"               //Velikost šířky dle obsahu
   android:layout_height="match_parent">             //Výška dle předka
   <TextView android:id="@+id/welcome_message"       //ID textu
      android:layout_width="wrap_content"            //Velikost šířky dle obsahu
      android:layout_height="wrap_content"           //Velikost výšky dle obsahu
      android:hint="@string/welcome_message"         //Obsah textu
      android:textStyle="bold"                       //Text bude tučně
      android:padding="20dp"                         //Vnitřní odsazení textu
      android:paddingTop="60dp"/>                    //Vnitřní horní odsazení textu
   <Button
      android:layout_width="wrap_content"            //Velikost šířky dle obsahu
      android:layout_height="wrap_content"           //Velikost výšky dle obsahu
      android:layout_centerInParent="true"           //Centruje vnitřek obsahu
      android:padding="20dp"                         //Vnitřní odsazení
      android:background="@drawable/mybutton" //Volitelné nastavení pozadí tlačítka
      android:text="@string/button_start"            //Text tlačítka
      android:onClick="startTestPressed" //Název akce, která se spustí při kliknutí
      android:layout_gravity="center" /> //Nastavuje zarovnání na střed
</RelativeLayout>
```

Ukázka 5 – První view

Proto bude využit relativní layout a zbytek vlastností se doladí pomocí stylů. Samotné view je k nalezení v res/layout/ s názvem hlavní Aktivity.

Existuje místo – v res/values/styles.xml, kde lze tyto styly sdružovat a pojmenovávat. Takové chování je dobré pro obecné a opakující se prvky stylu. V případě referenčního projektu je trochu zvláštní světlé pozadí a text je obecně dost malý. Ten se nabízí ke zvětšení, neboť v celé aplikaci textu bude stejně málo.

```
<resources>
   <style name="AppTheme" parent="Theme.AppCompat">  //Název a typ stylu
      <item name="android:textSize">20sp</item> //Zvětšení textu pro celou app
   </style>
</resources>
```

Ukázka 6 – Nastavení obecných stylů v styles.xml

Zbytek stylu se bude řešit lokálně, protože je individuální pro dané prvky viz ukázka 5. Zde je vidět, že se dá vzhled upravovat podobně jako s HTML a CSS u webů,

jen je třeba dávat pozor na jednotky velikosti a obecně se spíše vyhýbat přesně definovaným rozměrům z důvodů variability displejů viz podkapitola 1.3 Zobrazení.

V tuto chvíli je obsahem jen text Hello world! a nefunkční tlačítko. Jak bylo zmíněno dříve, nedoporučuje se z důvodu přehlednosti a udržitelnosti psát texty do kódu ať už view nebo jinam mimo res/values/strings.xml.

```
<resources>
    <string name="app_name">Download Speed Test</string>
    <string name="welcome_message">Vítejte v aplikaci ke zjištění Vaší rychlosti
připojení k síti!</string>
    <string name="button_start">Spustit test</string>
    /* Colors */
    <color name="black">#000000</color>
    <color name="white">#FFFFFF</color>
    <color name="dark_blue">#ff2a689e</color>
</resources>
```

Ukázka 7 – Seznam textů a barev ve strings.xml

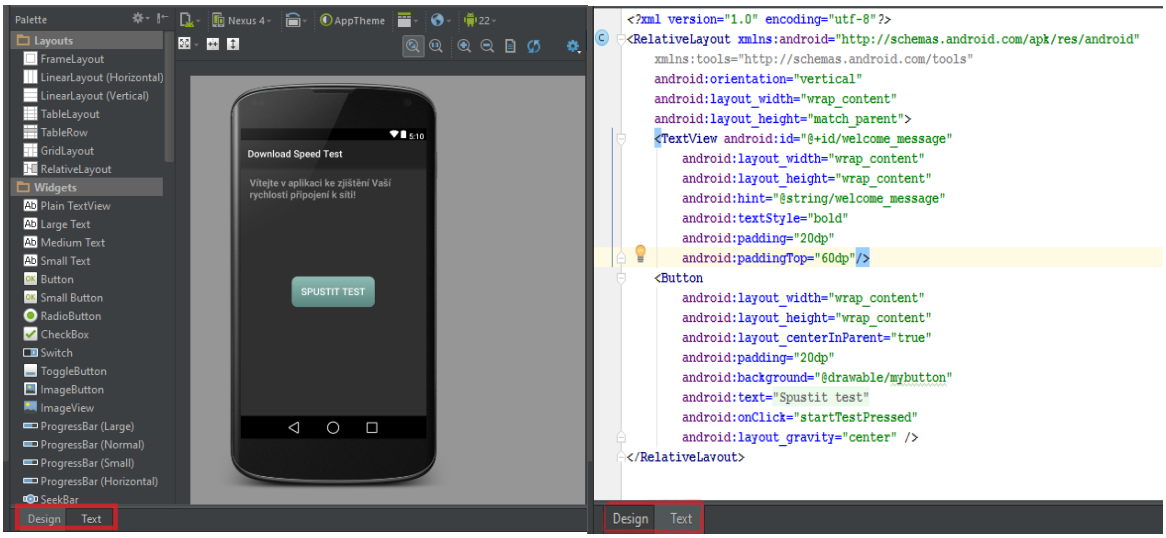
Není třeba zde vypisovat všechny přidané/upravené texty ze strings.xml, důležité je pochopit vazbu názvu řetězce s voláním konkrétního užití například textu tlačítka: android:text="@string/button_start" z view. Tím se zajišťuje párování textu s pozicí v aplikaci. V případě strings.xml je vidět, že i barvy mohou být také považovány za textové řetězce, proto je lze zde definovat a používat je podobným způsobem.

Nyní stačí zprovoznit tlačítko a první obrazovka bude hotová. Logicky je takové chování potřeba nastavit v příslušné Aktivitě. To je příkladně jednoduché, jen je třeba dopsat metodu s názvem korespondujícím s view android:onClick="startTestPressed:

```
public void startTestPressed(View view) {    isNetworkConnected(); }
```

Ukázka 8 – Metoda pouštějící se při stisku tlačítka

Zajímavé je, jak přívětivě spolupracuje prostředí Android Studia, mnoho těchto rutinních činností kontroluje a leckdy generuje potřebné kusy kódu samo. Např. při rozhodnutí využívat pro tvorbu view místo textové verze verzi designovou, lze view skládat pomocí přetahování drag and drop a tím šetřit čas hledáním syntaxí.



Obrázek 12: Úprava view v designové a textové podobě

Poslední drobnost spíše kosmetického charakteru stojící za to udělat, kvůli nepříliš valnému vzhledu tlačítka, je úprava pozadí. Tu je třeba nastavit v res/drawable, v tomto případě v mybutton.xml[23].

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android" >
   <item android:state_pressed="true" >
      <shape android:shape="rectangle"  >
         <corners android:radius="12dip" />
         <stroke android:width="1dip" android:color="#5e7974" />
         <gradient android:angle="-90" android:startColor="#345953" android:end-
Color="#689a92"  />
      </shape>
   </item>
   <item android:state_focused="true">
      <shape android:shape="rectangle"  >
         <corners android:radius="12dip" />
         <stroke android:width="1dip" android:color="#5e7974" />
         <solid android:color="#58857e"/>
      </shape>
   </item>
   <item >
      <shape android:shape="rectangle"  >
         <corners android:radius="12dip" />
         <stroke android:width="1dip" android:color="#5e7974" />
         <gradient android:angle="-90" android:startColor="#8dbab3" android:end-
Color="#58857e" />
```

[23] Protože v nastavení tlačítka ve view je android:background="@drawable/mybutton"

```
        </shape>
    </item>
</selector>
```

Ukázka 9 – Úprava pozadí tlačítka

Kód z ukázky vytvoří u tlačítka kulaté rohy a vybarví pozadí gradientem, viz obrázek 12. To demonstruje sofistikovanější grafické modifikace aplikace, což lze příhodně provést u mnohem širší palety prvků, než jen u tlačítka.

Tím se práce na první obrazovce uzavírá, ale Aktivita se bude ještě výrazněji měnit. Dle vysvětlení v 3.4 Rozvržení, další postup bude skrze samotné měření/dialogy a tedy nebude potřeba nové Aktivity/view.

3.7 Test připojení

V poslední podkapitole bylo nastaveno, že se bude volat při stisku tlačítka metoda isNetworkConnected(). To je z důvodu zajištění možnosti znovuvolání stejné funkce při volbě uživatele o opakování měření. Zde začíná samotná činnost aplikace, jíž je stažení souboru z předem nastaveného umístění na webovém serveru, zobrazení vývoje stahování do již existujícího pohledu a po ukončení stahování předání výsledku a započetí 2. části měření – odesílání. Je třeba implementovat třídy, které dokáží takovou činnost. Naštěstí Android Studio opět nabízí pomoc, kdy při napsání metody z této třídy automaticky aktivuje akci, jež Vám doplní skrze zkratku alt+enter neimportované potřebné prvků. Ostatně touto zkratkou se obecně řeší hodně nedostatků kódu.

```
private void isNetworkConnected() {
    ConnectivityManager cm = (ConnectivityManager)
        getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo ni = cm.getActiveNetworkInfo();
    if (ni!= null) {
        new TestConnectionAsync().execute();
    } else {
        Log.d("Internet Failure", "No network available!");
        showResult(getString(R.string.connection_warning));
    }
}
```

Ukázka 10 – Test síťového připojení skrze isNetworkConnected()

Teď již lze začít plnit isNetworkConnected() potřebným kódem. Nejdříve se zkontroluje připojení k síti, jak název napovídá. Dle vysvětlení v podkapitole 1.8 Komunikace s okolím tento test, byť doporučovaný samotným serverem Android Developer (4), prokazuje, že zařízení je připojené k nějaké síti. To ale nezaručuje přístup k internetu, neboť se s jistotou neví, zda je daná síť připojena na vnější síť.

Proto je zde nadstandardně přidán ještě test přímo na umístění budoucího souboru pro upload pro zjištění, zda je se aplikace schopna připojit ke kýženému serveru. To se děje skrze HttpUrlConnection a výsledek naplňuje proměnnou network. Protože je to operace, která by mohla zatížit hlavní vlákno, nelze ji tak použít. Je třeba ji vložit do AsyncTasku a tedy nového vlákna, na jehož konci se volá goToDownload().

```
try {
   HttpURLConnection urlc = (HttpURLConnection) (new URL("http://www.djha-
wky.cz/www/speedtest/UploadToServer.php").openConnection());
   urlc.setRequestProperty("User-Agent", "Test");
   urlc.setRequestProperty("Connection", "close");
   urlc.setConnectTimeout(1500);
   urlc.connect();
   if(urlc.getResponseCode() == 200){
      return true; //hodnota proměnné network
   } else{
   Log.d("Internet Failure:", "No network available!");
   }
} catch (IOException e) {
   Log.e("Internet Failure:", "Error checking internet connection", e);
}
return false;  //hodnota proměnné network
```

Ukázka 11 – Test internetového připojení pomocí AsyncTasku

Napříč aplikací se využívá vypisování chyb pomocí syntaxe Log.*. To je referencí na tzv Logcat, což je logovací zprávový systém, jenž se zobrazuje ve spodní části Android Studia pod záložkou logcat[24]. Má podobu konzole a díky filtraci typu zpráv se lze dobře orientovat, co se s aplikací/ přístrojem v danou chvíli děje. Proto se za Log píše písmeno typu určující charakter zprávy. Mezi typy patří d = debug, e = error, i = info ad.

[24] Defautní nastavení již logcat zobrazuje, při schování lze v nabídce opět povolit.

Při testování připojení byla v AsyncTasku upravena proměnná network dle výsledku testu. To je využito v rozhodovací struktuře, ale samotná proměnná není zatím definovaná, na což Android Stuido okamžitě upozorní. Proto se vkládá do hlavní Aktivity spolu s několika dalšími důležitými proměnnými její definice.

```
private ProgressDialog mProgressDialog;      //Proměnná pro první progress bar
private ProgressDialog mProgressDialog2;     //Proměnná pro druhý progress bar
private int timeLimit = 10000;        //Nastavení časového limitu toku dat (v ms)
private String downloadResult;        //Proměnná pro uchování výsledku stahování
private String uploadResult;          //Proměnná pro uchování výsledku odesílání
private boolean network = false;      //Proměnná pro rozhodnutí připojení k internetu
public static final int DIALOG_DOWNLOAD_PROGRESS = 0; //výchozí nastavení horizontálního progress baru
```

Ukázka 12 – Atributy v první aktivitě

3.8 Měření stahování

Dvě z nastavených proměnných používá již goToDownload(), jenž zároveň připravuje a spouští měření stahování, viz ukázka 13. Aby k takovému procesu došlo, musí se vyhodnotit proměnná network jako True tedy musí projít testem internetu. Poté se v metodě iniciuje první progress bar a AsyncTask na stažení souboru, jehož adresa je uvedena v myUrl. Ten má 100MB a webový server, na němž soubor leží, povoluje přístup k tomuto souboru. Dále je přiložena ukázka kódu již samotného stahování.

```
private void goToDownload() {
   if((network)) {
      String myUrl = "http://www.djhawky.cz/www/speedtest/100mb.test";
      new DownloadFileAsync().execute(myUrl);
      mProgressDialog = new ProgressDialog(this);
      mProgressDialog.setMessage(getString(R.string.downloading));
      mProgressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
      mProgressDialog.setCancelable(true);
      mProgressDialog.setProgress(DIALOG_DOWNLOAD_PROGRESS);
      mProgressDialog.show();
   } else {
      showResult(getString(R.string.internet_warning));
   }
}
```

Ukázka 13 – Nastavení pro spuštění stahování a kontrola stavu internetu v goToDownload()

```
try {
    URL url = new URL(aurl[0]);
    URLConnection conn = url.openConnection();
    conn.connect();
    File SdPath = Environment.getExternalStorageDirectory();
    File file = new File(SdPath + "/download/speedtest/test_down-
load_file.dat");
    InputStream input = new BufferedInputStream(url.openStream());
    OutputStream output = new FileOutputStream(file);
    byte data[] = new byte[1024];
    int count;
    long total = 0;
    long startTime = SystemClock.elapsedRealtime();
    while ((count = input.read(data)) != -1) {
        total += count;
        long dur = ((SystemClock.elapsedRealtime() - startTime));
        int percent = (int) ((dur)*100/ timeLimit);
        publishProgress(""+ percent);

        if(SystemClock.elapsedRealtime() - startTime >= timeLimit){
            break;
        }
    }
    double speed = (total*8/(SystemClock.elapsedRealtime() - startTime));
    double roundedSpeed = round(speed*1000/1024/1024,3);
    downloadResult = getString(R.string.speed_info_download) + " <font
color='#EE0000'>" + roundedSpeed +" Mb/s</font>\n";
    output.flush();
    output.close();
    input.close();
    try {
        file.delete();
    } catch (Exception e){
        return null;
    }
} catch (Exception e) {
    e.printStackTrace();
    Log.e("Got Exception :", "see logcat " + e.getMessage(), e);
}
return null;
```

Ukázka 14 – Jádro funkce měření stahování

Toto je základem celého programu, a proto stojí za to si ho přiblížit. V první části je spuštěn proces, který jsme v podobné obdobě viděli už při testu připojení. Na něj navazuje definice cesty, kam se bude ukládat stahovaný soubor. Zde je dobré mít na

zřeteli, že různá zařízení mají různou cestu k Download složce. Proto je použito Environment.getExternalStorageDirectory();, jenž zajistí navedení na správné místo.

Dále se nastaví stream pro přijetí a odeslání souboru, kde byla pro měření toku dat záměrně vybrána bufferová varianta, která je mnohem výhodnější než obecný InputStream viz podkapitola 2.2 Problematika měření. S tím souvisí nastavení dalších proměnných pod tím. Poté se definuje startTime s SystemClock.elapsedRealtime(), což je funkce podobná klasickému TimeStamp, ale pro Android. Dalším rozdílem je, že hodnota říká v milisekundách čas od posledního nastartování zařízení. Z tohoto důvodu lze porovnávat jen hodnoty elapsedRealtime mezi sebou, nelze jakkoliv pracovat s takovou hodnotou například v porovnání s časovou značkou serveru atp. Pro vnitřní účely aplikace je tato varianata naprosto dostačující. Časová značka je logický ukazatel, že v části ohraničené while probíhá samotné měření stahování.

While má podmíněné trvání do doby input.read(data))!=-1, než se stáhne soubor. Poté přičítá count aktuální obsah bufferu do celkového množství total, což dává informaci o tom, kolik již bylo přijato dat v bytech. Dále se vypočítává doba, po kterou běží proces přijímání dat, čehož se využívá v podmínce pro ukončení stahování na základě vypršení nastaveného časového limitu.

Prostor mezi těmito řádky slouží k ukazování stavu stahování. Zde byl rozhodnut kompromis, neboť je ukazování stavu v dané situaci fakticky problematičtější. Tím, že aplikace dopředu neví, zda se jí soubor povede stáhnout celý, nebo zda bude utnuta po časovém limitu, nelze s jistotou říct, co vlastně ukazovat. Původní výpočet na základě staženého souboru je zde, kde lenghtOfFile je celková velikost souboru:

```
publishProgress(""+(int)((total*100)/lenghtOfFile));
```

Ukázka 15 – Alternativní kód výpočtu stavu aplikace

To funguje dobře, když se podaří soubor stáhnout. Aby byl ovšem stažen do 10 sekund[25], musela by síť mít rychlost 80Mb/s, což je nepravděpodobné i u Wi-Fi sítě.

[25] Doba nastavená v private int timeLimit = 10000;

Protože se téměř jistě nepovede stáhnout celý soubor, ukazuje progress bar jen dílčí postup aplikace, aby ji v několika procentech ukončil. Proto se zdá rozumnější využít výpočet na základě uplynulého času děleným časovým limitem v procentech. To bude z uživatelského pohledu pravdivějším, smysluplnějším, více odpovídajícím stavem průběhu a na celkový výpočet rychlosti to nemá vliv.

Po ukončení while jsou známy všechny potřebné proměnné pro samotný výpočet rychlosti. Zde je potřeba si hlídat měrné jednotky, protože finální výsledek by bylo vhodné prezentovat v Mb/s. Technicky vzato též není potíž pro rychlosti v řádu kb/s vytvořit ještě jednu podmínku a nechat zvolit aplikaci, jakou měrnou jednotku použije. Při programování potřebného skriptu se navíc objevily dvě malé komplikace. První byla v limitu, který podporuje datový typ double. Ten přes velký rozsah podpory[26] nedokázal přesně a v jednom kroku pojmout početní operace nad rychlostí. Proto je výpočet rozložen na dvě části. Total *8 představuje celkový počet přesunutých dat převedených na bity. Dělí se časem trvání, což dává výsledek počtu bitů za počet milisekund. V druhé fázi se výsledek převádí na Mb/s a rovnou zaokrouhluje na 3 desetinná místa, což byla druhá drobná komplikace. Z toho důvodu vznikl jednoduchý skript, který dokáže zaokrouhlovat double na zvolený počet míst v druhém parametru:

```
public static double round(double value, int places) {
   if (places < 0) throw new IllegalArgumentException();
      BigDecimal bd = new BigDecimal(value);
      bd = bd.setScale(places, RoundingMode.HALF_UP);
      return bd.doubleValue();
}
```

Ukázka 16 – Funkce zaokrouhlování

Teď už je známa rychlost, ta se uloží do proměnné downloadResult spolu s výslednou částí řetězce[27], ukončí se oba streamy, vyčistí se paměť a smaže se případně nahraný soubor. Výhodou je, že ten je pravděpodobně prázdný, neboť dokud se nestáhne celý soubor, nenahrají se žádná data do souboru na disku.

[26] 2^-1074 <= x <= (2-2^-52)·2^1023, kde x je číslo double.

[27] Tento řetězec je napsán přímo v Aktivitě, protože nelze měnit vlastnosti textu s proměnnými uvnitř.

Při ukončení AsyncTasku se volá metoda onPostExecute(), která umí předat výsledek a lze s ní určit, kudy se má aplikace dále ubírat.

```
protected void onPostExecute(String result) {
   mProgressDialog.dismiss();
   goToUpload();
}
```

Ukázka 17 – onPostExecute u stahování

V tomto případě se ukončí činnost progress baru a volá se goToUpload(). Ten není zapotřebí ukazovat, protože má velmi podobnou strukturu jako gotToDownload(), jen nemusí rozhodovat o stavu sítě. Tím se uzavírá měření stahování.

3.9 Měření odesílání

Odesílání se může zdát v principu velmi podobné, ale lze jej řešit i jinak. Proto se tato podkapitola bude zabývat dvěma různými způsoby zpracování, odlišujícími se na základě přístupu měření.

- Případ 1 - Měření pomocí bufferu – neboli metoda použitá u stahování
- Případ 2 - Měření se zpracováním množství dat /čas na straně serveru

To zdánlivě může vést k myšlence, že první možnost se obejde bez komunikace se serverem, to ale není možné. Bez nastavení serveru a pomocného souboru na této straně nelze odesílat soubor na dané umístění. Celkový proces odesílání obsahuje fázi načtení souboru z umístění v zařízení, navázání komunikace se serverem a postupné odesílání dat a u případu 2 ještě jednou navázání kontaktu se serverem pro zjištění výsledku hodnot. Výpočet se bude konat na straně zařízení[28].

3.9.1 Případ 1

Případ jedna je dosti podobný stahování, budiž tedy zmíněny jen odlišnosti od předchozího použití.

[28] Výpočet lze dělat i na straně serveru a byla by to pravděpodobně z pohledu výkonu a hardwarového využití správnější volba. Tato práce se ale zabývá programováním v Androidu a ne v PHP.

V první řádě je třeba zvolit komunikaci se serverem, jaké on bude rozumět. V případě webového Apache PHP serveru se nejčastěji pro nahrávání souboru používá nějaká forma (nejen) formulářových metod POST nebo GET. Ty mají relativně hodně specifický a exaktní tvar, především se do hlavičky odesílaného souboru musí vyplnit potřebná data v daném formátu jako je vidět v ukázce níže. Mezi nimi je např. zakomponováno, jaká metoda se používá, samozřejmě název souboru a taktéž musí obsahovat informaci o tom, že využívá multipart/form-data metodu, standardně používanou pomocí HTML. Ze stejného důvodu je používána funkce writeBytes(), jež zařídí správný formát souboru.

```
protected String doInBackground(String... aurl) {
   int count;
   String upLoadServerUri = "http://www.djhawky.cz/www/speedtest/Upload-
ToServer.php";
   File SdPath = Environment.getExternalStorageDirectory();
   final String uploadFilePath = SdPath + "/download/speedtest/";
   final String uploadFileName = "10mb.test";
   String fileName = uploadFilePath + "" + uploadFileName;
   HttpURLConnection conn;
   DataOutputStream dos;
   String lineEnd = "\r\n";
   String twoHyphens = "--";
   String boundary = "*****";
   File sourceFile = new File(fileName);
   TelephonyManager tm = (TelephonyManager) getSystemService(Context.TE-
LEPHONY_SERVICE);
   String imei = tm.getDeviceId();
   String finalUploadFileName= imei + ".dat";

   if (!sourceFile.isFile()) {
      mProgressDialog2.dismiss();
      Log.e("uploadFile", "Source File not exist :"
            +uploadFilePath + "" + uploadFileName);

      runOnUiThread(new Runnable() {
         public void run() {
            uploadResult=getString(R.string.file_missing)
                  +uploadFilePath + "" + uploadFileName;
         }
      });
      return null;
   }
   else {
      try {
```

```
        Log.e("uploadFile", "Source File exist :" + uploadFilePath + "" + upload-
FileName);
        InputStream input = new BufferedInputStream(new FileInputStream(source-
File));
        URL url = new URL(upLoadServerUri);
        conn = (HttpURLConnection) url.openConnection();
        conn.setDoInput(true);
        conn.setDoOutput(true);
        conn.setUseCaches(false);
        conn.setChunkedStreamingMode(0);
        conn.setRequestMethod("POST");
        conn.setRequestProperty("Connection", "Keep-Alive");
        conn.setRequestProperty("ENCTYPE", "multipart/form-data");
        conn.setRequestProperty("Content-Type", "multipart/form-data;boun-
dary=" + boundary);
        conn.setRequestProperty("uploaded_file", finalUploadFileName);

        dos = new DataOutputStream(conn.getOutputStream());
        dos.writeBytes(twoHyphens + boundary + lineEnd);
        dos.writeBytes("Content-Disposition: form-data; name=\"uploa-
ded_file\";filename=\"" + finalUploadFileName + "\"" + lineEnd);
        dos.writeBytes(lineEnd);

        long startTime = SystemClock.elapsedRealtime();
        byte data[] = new byte[1024];
        long total = 0;

        while ((count = input.read(data)) != -1) {
           long dur = ((SystemClock.elapsedRealtime() - startTime));
           int percent = (int) ((dur)*100/timeLimit);
           publishProgress(""+ percent);
           total += count;
           dos.write(data, 0, count);

           if(SystemClock.elapsedRealtime() - startTime >= timeLimit){
              break;
           }
        }
         long endTime = SystemClock.elapsedRealtime();
        // Posílá povinná multipart form data po ukončení odesílání...
        dos.writeBytes(lineEnd);
        dos.writeBytes(twoHyphens + boundary + twoHyphens + lineEnd);

        double speed = (total*8/( endTime - startTime));
        double roundedSpeed = round(speed*1000/1024/1024,3);
        uploadResult = getString(R.string.speed_info_upload) + " <font
color='#EE0000'>" + roundedSpeed + " Mb/s</font>";
…atd
```

Ukázka 18 – Měření odesílání případ 1

U HttpURLConnection se objevují nové parametry za účelem povolení streamů a zakázání cache paměti. Stejně tak jako setChunkedStreamingMode(0); je doporučena serverem Android Developer (4) kvůli lepší efektivnosti úkolu. Ve stahování i odesílání je použit buffer 1024. Není to vyloženě podmínka, protože známe velikost souboru, lze zvolit i jinou velikost parametru. Jde spíše o adekvátně velký buffer v případě přerušení toku dat v závislosti na přesáhnutí časového limitu.

Jak je vidno ze skriptu, je změněn název souboru při ukládání. Pokud by zůstal původní název, aplikace by měla potíže při vyšším vytížení. Z logiky věci může nastat situace, kdy bude své soubory v jednu chvíli odesílat více uživatelů, což by mohlo ovlivnit pravdivost výsledku. V situaci jako tato ale stačí určit unikátní token, podle něhož se bude jmenovat cílový soubor a tím zamezit konfliktům dat. Jako ideální se, z charakteru své povahy, zdá být využití unikátního identifikačního čísla přístroje IMEI, jež lze získat skrze TelephonyManager a getDeviceId(). Všechny ostatní části jsou stejné se stahováním včetně již navržených změn a úvah.

3.9.2 Případ 2

Logika nabádá, že správnější a jistější je měření na straně příjemce. Proto je vhodné si ukázat i tuto možnost. To ale znamená, že po samotném odesílání se musí aplikace ještě zkontaktovat se serverem a získat výsledky měření. Pro přehlednější kód a zjednodušení byl nainstalován balík podporující komunikaci s Apache serverem - Apache HttpComponents. Ten má díky svým metodám elegantnější a méně obsáhlý kód viz ukázka níže.

Rozdíl je rychle znatelný. Je třeba mnohem méně proměnných, nastavení Multipart je mnohem kratší, stejně tak ubyly řádky spojené se samotným připojením a jeho uzavřením. Co přibylo je if sekce pro přijetí výsledku, která ale doopravdy začala již 2 řádky nad if sekcí.

Na serveru se množství dat změří na základě velikosti staženého souboru. Jeho velikost se při uploadu postupně navyšuje a v případě ukončení stahování obsahuje veškerá dosud stažená data. Technickou drobností, spojenou s přijetím dat, je, že výsledek se přijímá jako jeden velký string, čemuž je potřeba upravit formát na straně serveru. Poté se pomocí oddělovače opět na straně Androidu výsledek rozdělí na čas ukončení a velikost staženého souboru s převedením na čísla.

```
try {
    String fileName = Environment.getExternalStorageDirectory() + "/down-
load/speedtest/10mb.test ";
    String postReceiverUrl = "http://www.djhawky.cz/www/mp3/UploadToSer-
ver.php";
    TelephonyManager tm = (TelephonyManager) getSystemService(Context.TE-
LEPHONY_SERVICE);
    String imei = tm.getDeviceId();
    String finalUploadFileName= imei + ".dat";

    HttpClient httpClient = new DefaultHttpClient();// Nový HttpClient
    HttpPost httpPost = new HttpPost(postReceiverUrl); // Hlavička metody POST

    File file = new File(fileName);
    FileBody fileBody = new FileBody(file);
    MultipartEntity reqEntity = new MultipartEntity(HttpMultipartMode.BROW-
SER_COMPATIBLE);
    reqEntity.addPart("file", fileBody);
    reqEntity.addPart("time_limit", timeLimit);
    reqEntity.addPart("file_name", finalUploadFileName);
    httpPost.setEntity(reqEntity);

    // Vykoná HTTP post request
    HttpResponse response = httpClient.execute(httpPost);
    HttpEntity resEntity = response.getEntity();

    if (resEntity != null) {
        String responseStr = EntityUtils.toString(resEntity).trim();
        String[] separated = responseStr.split(",");
        Long duration = Long.parseLong(separated[0],10);
        Long lenghtOfFile = Long.parseLong(separated[1], 10);

        double speed = (lenghtOfFile*8/(duration));
        double roundedSpeed = round(speed*1000/1024/1024,3);
        uploadResult = getString(R.string.speed_info) + " <font color='#EE0000'>"
+ roundedSpeed + " Mb/s</font>";
    }

} catch(NumberFormatException nfe) {
    System.out.println("Could not parse " + nfe);
} catch (NullPointerException e) {
    e.printStackTrace();
} catch (Exception e) {
    e.printStackTrace();
}
mProgressDialog2.dismiss();
return null;
```

Ukázka 19 – Měření odesílání případ 2

Při zmíňce o čase je dobré zaregistrovat, že lze změřit startovní čas způsobem srovnatelným se serverem pomocí System.currentTimeMillis(). V případu 2 toho nebude využito, protože bude přesnější změřit startovní i konečný čas na druhé straně a přijmout pouze dobu trvání. Z toho důvodu se i časový limit předává pomocí hlavičky.

3.9.3 Serverová část

S ohledem na zadání této práce, není třeba do detailu rozebírat serverovou část aplikace. Je dost pravděpodobné, že při vývoji bude nastavovaný server jiného typu a bude využívat jiný programovací jazyk. Přesto budou některé prvky společné a je vhodné se o nich zmínit.

Při práci se souborem se doporučuje vytvořit složku, kam se budou jednotlivé soubory nahrávat[29]. Ty je pak potřeba po ukončení měření mazat, protože jinak budou u více uživatelských instancí zabírat více prostoru. V případu 1 není třeba nastavovat další funkce, u 2. varianty je struktura trochu složitější.

Nejdříve se přijmou proměnné časového limitu a názvu souboru, zapne se časomíra s přesností na milisekundy. Poté se při stahování musí kontrolovat, zda soubor nebyl stažen nebo zda se nepřesáhl časový limit. Na to se ukončí časomíra, vypočítá se z toho délka stahování a zjistí se velikost stažených dat. Pakliže se vytvořil fyzický soubor[30], je ho potřeba smazat, pakliže ne, smazat jej z paměti. Skript se celý zakončí navrácením výsledných hodnot 2 proměnných. Délka stahování a velikosti stažených dat se vloží do jednoho řetězce v tomto pořadí a oddělí se čárkou.

3.10 Odeslání hodnot

Ať je zvolena první nebo druhá varianta měření odesílání, na konci AsyncTasku jsou již známé obě potřebné hodnoty. Spustí se příkazy pro ukončení druhého progress baru, spojí se výsledek stahování a odesílání do konečného řetězce a tento je poslán skrze metodu showResult(). Ta již byla několikrát ve skriptech uvedena a její funkcí je předání výsledku měření nové Aktivitě, kterou iniciuje. Nezáleží na tom, zda aplikace

[29] Složce je třeba dát potřebná oprávnění na zápis.

[30] Např. PHP servery nejdříve příchozí soubory ukládají do paměti, bez zpracování zůstávají v $_FILES.

prošla celým měřením a má výsledky nebo narazila na jednu z překážek v podobě testu připojení. Aplikace by ve výsledku měla informovat skrze showResult o výsledku.

Na ukázce si je dobré všimnout syntaxe, jak pojmenovat a předat proměnou nové Aktivitě – intent.putExtra(). Stejný řetězec se pak použije dále.

Tímto končí práce první Aktivity, která se dostává do stavu onStop() dle Obrázek 1: Graf životního cyklu Aktivity , nicméně je stále spuštěna.

```
private void showResult(String result) {
   Intent intent = new Intent(this, ResultActivity.class);
   intent.putExtra("EXTRA_SPEED_INFO", result);
   startActivity(intent);
}
```

Ukázka 20 – metoda volání nové Aktivity s předáním výsledku měření

3.11 Dokončení aplikace

Při výběru možnosti v menu File/New/Activity se vygeneruje povinná struktura Aktivity včetně základních metod a Aktivita se zapíše do Manifestu. V tomto kódu se budou dít změny jen v rámci metody onCreate(). Zde se nyní nachází jen kód super.onCreate(savedInstanceState)[31] a nastavení, které view se má použít. Jako první k tomu přibyde získání výsledku měření z intentu:

```
   Intent myIntent = getIntent();
   String speedInfo = myIntent.getStringExtra("EXTRA_SPEED_INFO");
```

Ukázka 21 – Přijetí výsledku měření v Aktivitě 2

Na to navazuje část dialogová, kde je cílem zobrazit výsledek a dát uživateli šanci měření opakovat nebo aplikaci zavřít. K tomu se váže problematika, jak fakticky zpracovat obě akce programu. Při opakování testu se zavře současná Aktivita a bude zapotřebí znovu obnovit původní Aktivitu spuštěním metody isNetworkConnected(). To se provádí stejně jako většina pohybu mezi Aktivitami skrze nový Intent.

[31] Zajišťuje volání kódu předků

V této situaci se nastaví značka - flag setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP), což dává informaci API, že kýženým záměrem je použít původní Aktivitu, jež se přenačte. Je totiž zbytečné volat novou Aktivitu, když je stále spuštěna ta původní (byť na pozadí a neaktivní). Stačí nastavit Intent s parametrem Start = True a původní Aktivita je připravena.

Při volbě ukončení je situace podobná, přestože jde o jinou akci. Zavření současné aktivity se zajistí pomocí finish(). Ta ale neukončí původní Aktivitu, a proto se zavolá původní Aktivita s Intentem a parametrem Exit = True.

```
   AlertDialog.Builder builder1 = new AlertDialog.Builder(this);
   builder1.setMessage(Html.fromHtml(speedInfo));
   builder1.setCancelable(true);
   builder1.setPositiveButton(getString(R.string.repeat_test),
         new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
               finish();
               Intent intent = new Intent(ResultActivity.this, ProcessingActi-
vity.class);
               intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
               intent.putExtra("Start", true);
               startActivity(intent); }});
   builder1.setNegativeButton(getString(R.string.close),
         new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
               Intent intent = new Intent(getApplicationContext(), ProcessingActi-
vity.class);
               intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
               intent.putExtra("Exit", true);
               startActivity(intent); }});

   AlertDialog alert11 = builder1.create();
   alert11.show();
```

Ukázka 22 – Zobrazení výsledku v Aktivitě 2

```
if(getIntent().getBooleanExtra("Start", false)){
   isNetworkConnected();
}
if (getIntent().getBooleanExtra("Exit", false)) {
   finish();
}
```

Ukázka 23 – Rozhodování v Aktivitě 1 na základě volby uživatele

Předané parametry intentu nejsou nic jiného než proměnné s názvy Start a Exit. Proto se do inicializace hlavní Aktivity (do metody onCreate()) vloží kód pro sledování, zda nebyla provedena volba z druhé Aktivity a jaký je její výsledek. Poté se buď Aktivita uzavře, nebo se volá isNetworkConnected(), což nastartuje nové měření. Tím se uzavře kruh akcí a aplikace je hotova.

3.12 Závěr praktické části

Aplikaci, tak jak je teď vytvořena, lze vytknout pár drobných nedostatků. Například příliš velké množství kódu v první Aktivitě není správný programátorský přístup. Jistě se dá oddělit samotná Aktivita a pro každý AsyncTask vytvořit vlastní samostatnou třídu mimo Aktivitu. To by znamenalo přeposílat proměnné v atributech volacích metod, ale ulehčením je, že progress bar lze nastavit přímo v AsyncTasku v metodě onPreExecute(). Ta funguje podobně jako onCreate() pro Aktivitu. Dále by bylo záhodno udělat Aktivitu obecných funkcí jako zaokrouhlování, kterou by pak samostatný AsyncTask nebo jiná třída mohla volat. Kód se stane členitější a logičtější a v žádném ze souborů se nepřesáhne víc jak 150 řádek kódu.

Další nereflektovanou potíží je stabilita serverové strany. V případě referenčního projektu se využívá obyčejný web server. Ten je sice zatím pro použití v malém měřítku schopný měření pojmout, ale při vícenásobném simultánním vytížení není rozhodně dostačující. Stejně tak je třeba přemýšlet i nad možnostmi podobné aplikace, neboť testování jedním člověkem přináší úplně jiné výsledky, než při mnohočetném zatížení. I když např. nebude server zcela vyřazen, vyšší aktivita může způsobit zpomalení jednotlivých měření a tím ovlivnit výsledek. S tím se pojí i myšlenka, že je rozhodně nutné najít server, jenž spolehlivě operuje s daty a přístupem k internetu v dostatečné šíři a rychlosti.

Taktéž grafika by mohla být vylepšena, především prezentace výsledků a průběhu by se mohla výrazněji vyzdobit například číselníky ve více atraktivním tvaru. Ty by se daly spojit v jeden velký progress bar s hodnotami pod sebou. To ale není smyslem práce, výrazně by se tím zkomplikovalo vysvětlování základních principů a charakterem projektu bylo spíše vytvoření přenositelného modulu pro činnost komplexnější aplikace, kde se GUI bude řešit pravděpodobně zvlášť.

I použití AsyncTasku je na hraně. Jak bylo zmíněno, AsyncTask se hodí na jednodušší činnosti do několika sekund. 10 a více sekund je již na hraně a některá reálná měření stahování obsahovala malou prodlevu při přechodu do odesílací fáze. To neovlivňilo výsledek měření, a proto s ohledem na jednodušší vysvětlitelnost a upevnění základů nebylo řešeno.

Svůj účel aplikace splňuje a zdá se, že i v dostatečně přesné míře v poměru k uživatelské přívětivosti.

Shrnutí výsledků

V úvodní části byla obecně zodpovězena otázka, proč se snažit programovat pro Android prostředí a jaké jsou možnosti prosazení se. Android je rozhodně platforma, která má velké zastoupení a dá se skrze ni oslovit obrovské množství lidí.

Programování v Androidu má svá specifika. Při hlubším proniknutí do základů frameworku vyvstává hodně otázek ohledně správného zpracování kýžených funkcí aplikace. Android je v tomto ohledu dobře vybaven, a i když logika komunikace se zařízeními má vlastní podobu, je přehledná a pochopitelná i díky dobře zpracovanému tutoriálu a dokumentaci.

Dále se práce zabývá metodikou měření, která ukázala, že je více cest, jak k výsledku dospět. Ke každé z nich jsou vázány nevýhody, záleží tedy na úvaze a pojetí, s jakými parametry a kompromisy bude program spjat. Byla zde vybrána metoda později použitá v praktické části.

Ta ukázala, že kód psaný v Androidu je přehledný, funkce dobře popsané a aplikace se dá vytvořit relativně efektivně, jakmile programátor pronikne do mechanik Androidu a API Android Studia. To je velmi nápomocné a výrazně pomáhá při rutinních činnostech, ale i v případě nápovědy možností nebo hlídání hrubých chyb. Samotná aplikace pak postupně pomocí jednotlivých podkapitol naplňuje svůj smysl a drží se plánu jak zvolené metody, tak doporučení navržených v teoretické části základu programování.

Aplikaci lze rozhodně vylepšit, nicméně tak jak je navržena a popsána i s ohledem na uživatelskou přívětivost, plní svůj úkol.

Závěry a doporučení

Platforma Android je na velkém vzestupu. Stále přibývá mnoho nových uživatelů, tím pádem i poptávka po aplikacích. Proto má téma práce smysl, neboť bude přibývat lidí, kteří budou chtít programovat v tomto prostředí. Práce přináší základní úvod do světa plného drobných pomocných projektů, z nichž se dá ucelením získat velmi silný a schopný nástroj pro každodenní činnosti i pro komplexnější řešení situací na základě rozličných podnětů.

Cílem této práce je přiblížit prostředí mobilních aplikací a Androidu programátorům, kteří mají malou nebo žádnou zkušenost. Ukazuje jim cestu a základní principy programování. Snaží se podávat informace srozumitelně od nejzákladnějších po podrobnější, ne však příliš do hloubky. To může být první výtka, protože tím se práce omezuje na dost malý okruh čtenářů, pro které je vhodná. Její nekomplikovanost je zároveň výhodou i nedostatkem.

Výběr metody měření přinesl hodně otázek, jaká je ideální forma měření i s ohledem na uživatele a přesnost dat. Metoda se musí vypořádat se situací, že mobilní přístroje mají více variabilní možnosti připojení, stejně tak i rychlost toku dat může nabývat velmi různých hodnot. Navíc stabilita vzhledem k bezdrátové technologii připojení výrazně ovlivňuje relevanci výsledků a nabádá k myšlence, co je vlastně chtěnou měřenou hodnotou s nejvíce vypovídající hodnotou. Neexistuje tedy jednoznačné a prosté řešení.

Samotná aplikace jde cíleně na začátku od jednoduššího a esenciálního k složitějšímu, což výrazně pomáhá s postupným osvojováním technik. Řešení problému se zdá být ale krkolomné. Jistě se dá aplikace napsat elegantněji a více přehledně, je však otázka, zda by byla stále tak lehce pochopitelná, jako je teď.

Program také může obsahovat nedostatky, ačkoliv byly některé ochranné mechanismy naimplementovány. Aplikace je hodně závislá na serverovém prostředí, k němuž se připojuje. To může být výrazným problémem měření v případě nepříliš stabilního umístění s malou průchodností.

Samotné téma je ovšem vybrané adekvátně, s ohledem na přiměřený level obtížnosti a dokázalo tak efektivně ukázat pravidla vývoje. Výběr prostředí Android Studia k tomu jednoznačně přispívá a celkově práce ukazuje pohodlnost a jednoduchost programování v Androidu v nekomplikovaných aplikacích.

Seznam literatury

1. **androidrank.org.** Android Market App Statistics. *ANDROIDRANK - open market statistics since 2011.* [Online] [Cited: 7 2, 2015.] http://www.androidrank.org/categorystats?category=&price=all.

2. **Gartner inc.** Usage share of operating systems. *Wikipedia.* [Online] 2. 7 2015. https://en.wikipedia.org/wiki/Usage_share_of_operating_systems.

3. **Konečný, Matěj - Zdroják.cz.** Vyvíjíme pro Android: První krůčky. *Zdroják.cz.* [Online] 22. 6 2012. http://www.zdrojak.cz/clanky/vyvijime-pro-android-prvni-krucky/.

4. **Android.** Application Fundamentals. *Developer Android .* [Online] 24. 05 2015. http://developer.android.com/guide/components/fundamentals.html.

5. **Herong, Yang Dr.** Introduction of Activity Lifecycle. *Android Tutorials - Herong's Tutorial Examples.* [Online] 2012. [Citace: 09. 07 2015.] http://www.herongyang.com/Android/Activity-Introduction-of-Activity-Lifecycle.html.

6. **Feltl, Michal - Svět Androida.** 2. díl - Různé density a rozlišení Android zařízení. *Svět Androida.* [Online] 11. 06 2011. [Citace: 11. 07 2015.] http://www.svetandroida.cz/2-dil-ruzne-density-a-rozliseni-android-zarizeni-201106.

7. **Android.** Introduction to Android. *Android Developer.* [Online] 24. 6 2015. http://developer.android.com/guide/.

8. **Techotopia.** Understanding Android Views, View Groups and Layouts. *Techotopia.* [Online] 01. 07 2013. [Citace: 06. 08 2015.]

http://www.techotopia.com/index.php/Understanding_Android_Views,_View_Groups_and_Layouts.

9. **Kumar, Praveen - Software And Technology Education.** Android Basics And Programming Tutorial. *Software And Technology Education.* [Online] 16. 4 2014. [Citace: 26. 9 2015.] http://speechustutorial.blogspot.cz/2014/04/android-tutorial.html.

10. **Apache.** Apache HttpComponents - Apache HttpComponents. *Apache.* [Online] The Apache Software Foundation, 23. 09 2015. [Citace: 01. 10 2015.] https://hc.apache.org/.

11. **Vrbacký, Jakub - mobilizujeme.cz.** Technologie mobilního internetu – od CSD po LTE Advanced (vědecké okénko). *Mobilizujeme.* [Online] 12. 02 2012. [Citace: 05. 07 2015.] http://mobilizujeme.cz/clanky/technologie-mobilniho-internetu-od-csd-po-lte-advanced-vedecke-okenko/.

12. **Macháček, Miloslav - Internet pro všechny.** Změřili jsme parametry sítě LTE O2. *Internet pro všechny.* [Online] 30. 3 2015. http://www.internetprovsechny.cz/zmerili-jsme-parametry-site-lte-o2/.

13. **InetDaemon.Com.** TCP 3-Way Handshake (SYN,SYN-ACK,ACK). *InetDaemon.Com - Free Online IT Tutorials and Internet Training.* [Online] 01. 09 2013. [Citace: 12. 09 2015.] http://www.inetdaemon.com/tutorials/internet/tcp/3-way_handshake.shtml.

Přílohy

Seznam obrázků

Obrázek 1: Graf životního cyklu Aktivity (5) 6
Obrázek 2: Rozdělení mobilních přístrojů podle zobrazení na 4 typy (7) 8
Obrázek 3: Příklad struktury více ViewGroup a View (7) 9
Obrázek 4: Vizualizace možné hierarchie s layout parametry (4) 9
Obrázek 5: Příklad skládání layoutů (8) 10

Obrázek 6: Anatomie Android aplikace 12
Obrázek 7: Příklady uživ. rozhraní (zleva action bar, dialog a status notification) zdroj (4) 15
Obrázek 8: Tabulka rychlostí bezdrátových mobilních připojení (11) 19
Obrázek 9: Stavový diagram aplikace 25
Obrázek 10: Ukázky obrazovek aplikace 25
Obrázek 11: Podrobný diagram aktivity aplikace 28
Obrázek 12: Úprava view v designové a textové podobě 31

Seznam tabulek

Tabulka 1: Počet Android aplikací dle počtu stažení (1) 2
Tabulka 2: Rošířenost operačních systémů na světě (2) 2
Tabulka 3: Porovnání metod měření 23

Seznam ukázek

Ukázka 1 – Volání intentu (4) 5
Ukázka 2 – Volání dle ID v závislosti na umístění 15
Ukázka 3 – Uses-permission v Manifestu 26
Ukázka 4 – Nastavení aplikace a Aktivit v Manifestu 27
Ukázka 5 – První view 29
Ukázka 6 – Nastavení obecných stylů v styles.xml 29
Ukázka 7 – Seznam textů a barev ve strings.xml 30
Ukázka 8 – Metoda pouštějící se při stisku tlačítka 30
Ukázka 9 – Úprava pozadí tlačítka 32
Ukázka 10 – Test síťového připojení skrze isNetworkConnected() 32
Ukázka 11 – Test internetového připojení pomocí AsyncTasku 33
Ukázka 12 – Atributy v první aktivitě 34
Ukázka 13 – Nastavení pro spuštění stahování a kontrola stavu internetu v goToDownload() 34
Ukázka 14 – Jádro funkce měření stahování 35
Ukázka 15 – Alternativní kód výpočtu stavu aplikace 36

Ukázka 16 – Funkce zaokrouhlování 37
Ukázka 17 – onPostExecute u stahování 38
Ukázka 18 – Měření odesílání případ 1 40
Ukázka 19 – Měření odesílání případ 2 42
Ukázka 20 – metoda volání nové Aktivity s předáním výsledku měření 44
Ukázka 21 – Přijetí výsledku měření v Aktivitě 2 44
Ukázka 22 – Zobrazení výsledku v Aktivitě 2 45
Ukázka 23 – Rozhodování v Aktivitě 1 na základě volby uživatele 45

Zadání práce

Univerzita Hradec Králové
Faculty of Informatics and Management
Akademický rok: 2015/2016

Studijní program: Applied Informatics
Forma: Combined
Obor/komb.: Aplikovaná informatika (ai2-k)

Podklad pro zadání DIPLOMOVÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Bc. Jestřáb Josef Jakub	Ocelíkova 672/1, Praha	I1300255

TÉMA ČESKY:

Vývoj mobilní aplikace pro android na testování internetového připojení

TÉMA ANGLICKY:

Development of Android application for testing of internet connection

VEDOUCÍ PRÁCE:

doc. Ing. Filip Malý, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl práce: Představení programování pro začínající programátory v prostředí Androidu s vybranou ukázkou na konkrétním použitelném programu pro měření internetového připojení.

Osnova práce:
-Úvod
-Základy mobilní aplikace
-Měření připojení
-Praktická část
-Shrnutí výsledků, závěry
-Literatura

SEZNAM DOPORUČENÉ LITERATURY:

Podpis studenta: ..... **Datum:** 11.11.

Podpis vedoucího práce: ..... **Datum:** 10.11

(c) IS/STAG , Portál - Podklad kvalifikační práce , I1300255 , 10.11.2015 11:50