Mendelova univerzita v Brně
Provozně ekonomická fakulta

RSS čtečka relevantních zdrojů orgánů veřejné správy

Bakalářská práce

Vedoucí práce:
Ing. Jan Kolomazník

Jan Jacko

Brno 2015

Poděkování

Na tomto místě bych rád poděkoval panu Ing. Janu Kolomazníkovi, vedoucímu mé bakalářské práce, za odborné vedení, cenné rady, připomínky a podkladové materiály, které mi ochotně poskytoval. Dále děkuji své rodině za podporu během studia a všem, kteří si našli čas k vyplnění dotazníku a poskytli mi tak cenné informace, ze kterých jsem vycházel při zpracování práce.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **RSS čtečka relevantních zdrojů orgánů veřejné správy** vypracoval samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*

Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 2. ledna 2015 _____

Abstract

Jacko, J. RSS relevant resources reader of public authorities. Bachelor thesis. Brno: Mendel University, 2015.
First of all this bachelor thesis determines users preferences by a questionnaire survey and use them during development of RSS reader for Android platform. Also marketing strategy for distributing the finished Android application will be based on the results of questionnaire survey. To obtain information that is not available through RSS feeds, will be designed and created API based on REST architecture. The main results are summarized in the conclusion of this thesis.

Keywords

Android, RSS reader, REST API, mobile app, tablet, smartphone, java.

Abstrakt

Jacko, J. RSS čtečka relevantních zdrojů orgánů veřejné správy. Bakalářská práce. Brno: Mendelova univerzita v Brně, 2015.
Bakalářská práce nejdříve identifikuje preference uživatelů za pomoci dotazníkového šetření, které následně využije během vývoje RSS čtečky pro Android platformu. Také marketingová strategie pro distribuci hotové Android aplikace bude vycházet z výsledků dotazníkového šetření. Pro získání informací, které nejsou dostupné prostřednictvím RSS kanálů, bude navrženo a vytvořeno vlastní API postavené na architektuře REST. Výsledky a doporučení jsou shrnuty v závěru bakalářské práce.

Klíčová slova

Android, RSS čtečka, REST API, mobilní aplikace, tablet, chytrý telefon, java.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod a cíl práce

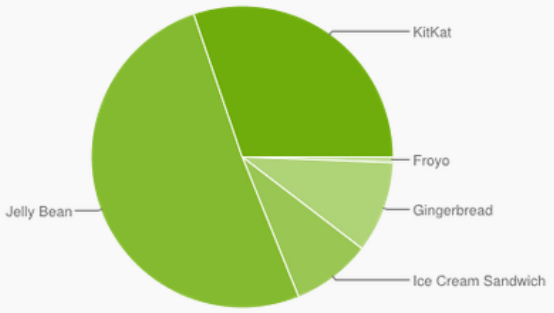
1.1 Android, nejoblíbenější mobilní platforma

Android pohání více než miliardu mobilních zařízení po celém světě a každý den se tento počet zvyšuje o více než 1 milion nových uživatelů. Ti stáhnou každý měsíc více jak 1,5 miliardy aplikací a toto číslo stále roste. Nabízí se tedy otázka, proč je tato platforma tak oblíbená? Důvodů je hned několik počínaje otevřeným zdrojovým kódem a snadnou modifikací systému, podporou různých procesorových architektur a konče obrovským výběrem aplikací a digitálního obsahu. (Android, the world's most popular mobile platform, 2011)

Jedinou komplikací je právě onen otevřený zdrojový kód a s tím spojené problémy. Společnost Google, která zaštituje vývoj této platformy, vydává minimálně jednou ročně aktualizaci přinášející různá bezpečnostní vylepšení a nové funkcionality. Ty se obvykle dostanou jen na zařízení s nijak nemodifikovaným operačním systémem. Uživatelé ostatních zařízení, kterých je drtivá většina, obdrží takové aktualizace, buď s velkým časovým zpožděním anebo dokonce vůbec. A právě to je příčinou současné situace, kdy jsou na trhu různé verze Androidu napříč zařízeními a mobilní vývojáři se s tím tedy musí nějakým způsobem vypořádat. Nově vyvíjené aplikace obvykle podporují i starší verze systému, protože jejich postavení na trhu je pořád významné.

Version	Codename	API	Distribution
2.2	Froyo	8	0.6%
2.3.3 - 2.3.7	Gingerbread	10	9.8%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	8.5%
4.1.x	Jelly Bean	16	22.8%
4.2.x		17	20.8%
4.3		18	7.3%
4.4	KitKat	19	30.2%

Data collected during a 7-day period ending on November 3, 2014.
Any versions with less than 0.1% distribution are not shown.



Obr. 1 Zastoupení jednotlivých verzí operačního systému Android
Zdroj: Dashboards: Platform Versions, 2011.

1.2 Cíl práce

Cílem práce je tedy vytvoření mobilní aplikace pro operační systém Android kompatibilní s verzemi 2.2 Froyo až 4.4 KitKat. Aplikace bude volně dostupná a bude primárně poskytovat podstatné informace z vybraných internetových stránek veřejné správy. Kvůli informacím, které nejsou dostupné ve formě klasického RSS

kanálu, bude potřeba vyvinout podpůrné webové rozhraní tzv. API. Aplikace také dokáže přehrávat aktuální videa z jednání Poslanecké sněmovny, spravovat libovolné RSS kanály, upozornit na události v podobě notifikace a mnoho dalších užitečných funkcionalit.

Výběr informačních zdrojů bude proveden za pomoci dotazníkového šetření a následné analýzy, takovým způsobem, aby co nejvíce vyhovoval preferencím uživatelů. Poté co vyřešíme tuto otázku, si představíme současné technologie vývoje aplikací pro Android platformu. A zjistíme, jaké možnosti máme na výběr, pokud chceme zaručit stejný vzhled aplikace i na starších verzích Androidu, optimalizovat funkčnost aplikace pro tablety či přehrávat videa v systémem nepodporovaném formátu. Díky získaným poznatkům navrhneme a implementujeme výslednou aplikaci a ověříme její funkčnost v praxi. A na závěr zhodnotíme její přínos, případně navrhneme další rozšíření.

1.3 Přehled literatury

Podle mého názoru panuje aktuálně na českém trhu relativní nedostatek publikací zabývajících se danou oblastí, proto jsem se při výběru literatury poohlédl i po zahraničních autorech. Z těch mě nejvíce zaujala publikace od Burtona a Felkera (2012), kteří opravdu srozumitelně a stručně vysvětlují jak začít s vývojem mobilních aplikací. V podobném duchu se nese i publikace Herouta (2010), která se zabývá základy jazyka Java a v začátcích také hodně pomůže. Oproti tomu Grant (2013) jde do daleko větších podrobností a díky jeho ukázkám kódu se dají vytvořit opravdu různé užitečné věci. Srovnatelné s ním jsou i Murphy (2011) nebo Ujbányai (2011). Ovšem žádná z uvedených publikací se nevyrovná oficiální dokumentaci Android Developers Guide (2011). Člověk zde nalezne kromě klasického popisu funkcí, datových typů a podobných informací také ukázkové příklady, doporučení a mnoho dalších užitečných tipů. Zcela bezesporu se jedná o nejvhodnější zdroj informací.

1.4 Metodika zpracování

Vývoj Android aplikací obecně probíhá ve třech navazujících krocích (Get Started, 2011):

1. Design

   Předtím než začneme psát jednotlivé řádky kódu, potřebujeme navrhnout uživatelské rozhraní splňující standardy a doporučení tvůrců Androidu. Přestože už tušíme, jaké operace bude uživatel provádět, měli bychom se zaměřit na způsob, jakým bude probíhat interakce. Z toho důvodu by měl být design funkční, jednoduchý a šitý na míru konkrétní aplikaci.

2. Vývoj

   Jakmile máme design hotový, potřebujeme ho převést do podoby reálné fungující aplikace. Android framework poskytuje všechna potřebná API k tvorbě

aplikací využívajících veškerých výhod hardwaru zařízení, připojeného příslušenství, internetové sítě a mnoho dalších.

3. Distribuce

   Naše aplikace je kompletní, podporuje širokou paletu zařízení s odlišnou velikostí displeje a hustotou pixelů. Je otestována na Android emulátoru i na skutečných zařízeních, a zbavena všech chyb. Jak budeme pokračovat, záleží na různých faktorech, jakými mohou být např. marketingová strategie a podporovaná škála mobilních zařízení.

2 Dotazníkové šetření

V tento okamžik kdy máme definovaný cíl a víme co, by měla Android aplikace umět, je potřeba provést analýzu požadavků budoucích uživatelů. Analýza bude provedena metodou dotazníkového šetření z důvodu nízké časové i finanční náročnosti. Tato metoda zajistí respondentům anonymitu a její výsledky půjdou také snadno vyhodnotit.

2.1 Příprava

Dotazník se bude týkat relevantních informačních zdrojů veřejné správy, které bude aplikace implicitně zobrazovat a zpracovávat. Jednodušeji řečeno budeme zkoumat, jaké oblasti veřejné správy zajímají samotné uživatele a rádi by je viděli v nově vytvořené aplikaci. Ovšem termín veřejná správa je velice široký pojem zahrnující mnoho subjektů, proto zúžíme výběr na státní správu a její ústřední orgány s celostátní působností. (Volek a Přenosil, 2005) Tento krok pomůže přehlednosti celé aplikace a zároveň výrazným způsobem respondentovi usnadníme proces vyplňování. Ostatní zdroje, jako RSS kanály jednotlivých krajů a obcí nebo jakékoliv další, si uživatel bude moci podle vlastního uvážení přidat sám v sekci aplikace k tomu určené.

Obecně vzato sestavování jakéhokoliv tzv. reprezentativního vzorku tak, aby nezkresloval výsledek šetření, je věc velice obtížná. Proto si tedy pojďme zdůvodnit volbu aplikovaných kritérií, podle nichž jsme cílovou skupinu lidí vybírali. Pokud bychom se podívali na uživatelskou základnu Android aplikací podobného zaměření, tzn. populární RSS čtečky typu *Feedly*, tak zjistíme, že jejich věkové složení se víceméně shoduje se skupinou všech uživatelů chytrých zařízení. (Feedly. Your news reader, 2012) To znamená, že naši cílovou skupinu uživatelů budou tvořit lidé ve věku od patnácti do padesáti let, kteří vlastní alespoň jedno chytré zařízení s operačním systémem Android a je jedno zda se jedná o telefon či tablet.

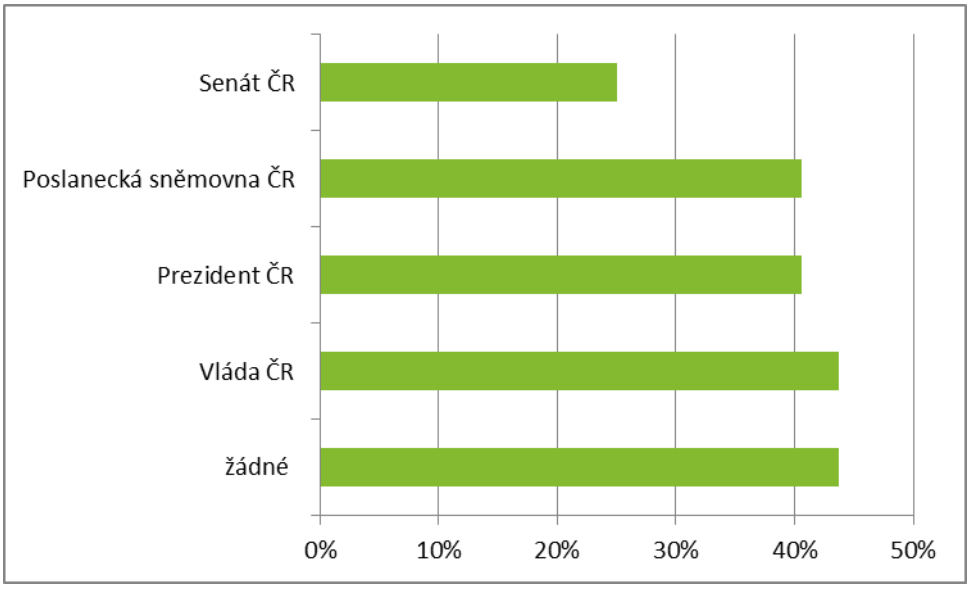
Při výběru jaký způsob sběru odpovědí zvolit, bude pro nás největší prioritou komfort samotného respondenta, tzn. snadný způsob výběru odpovědí a také abychom s ním oslovili co největší počet zájemců, kteří jsou ochotni se do průzkumu zapojit. Na výběr máme k dispozici následující varianty sběru dat: ústní, telefonické a či běžné zaškrtávání odpovědí. Nejschůdnější variantou se zdá být poslední jmenovaná. Avšak kvůli nákladům spojeným s tiskem velkého počtu stránek se rozhodneme pro ekologicky šetrnější elektronickou variantu. Jmenovitě tedy Formuláře Google, jenž budou našim potřebám dostatečně vyhovovat.

Konkrétní znění otázek se nachází v příloze A. K nim je však nutné si říci, proč jsme si je formulovali zrovna v této podobě. Je zřejmé, že otázky jsou polytomického typu. U nich je nutno dát si pozor při rozhodování, zda zařadit i neutrální možnost jako třeba nevím, aby nebyla odpověď zkreslená v případě, že uživatel opravdu nezná odpověď. Vzhledem k speciálnímu zaměření našich otázek tedy takovou variantu odpovědi povolíme, neboť ne každý se zajímá o toto téma. U druhé otázky má ze stejného důvodu respondent dokonce možnost odpovědět vlastními slovy

a sdělit nám svůj postoj. Tím pádem máme dotazník již připraven a můžeme začít se sběrem odpovědí, který tedy bude probíhat umístěním dotazníku na sociální sítě.

2.2 Vyhodnocení odpovědí

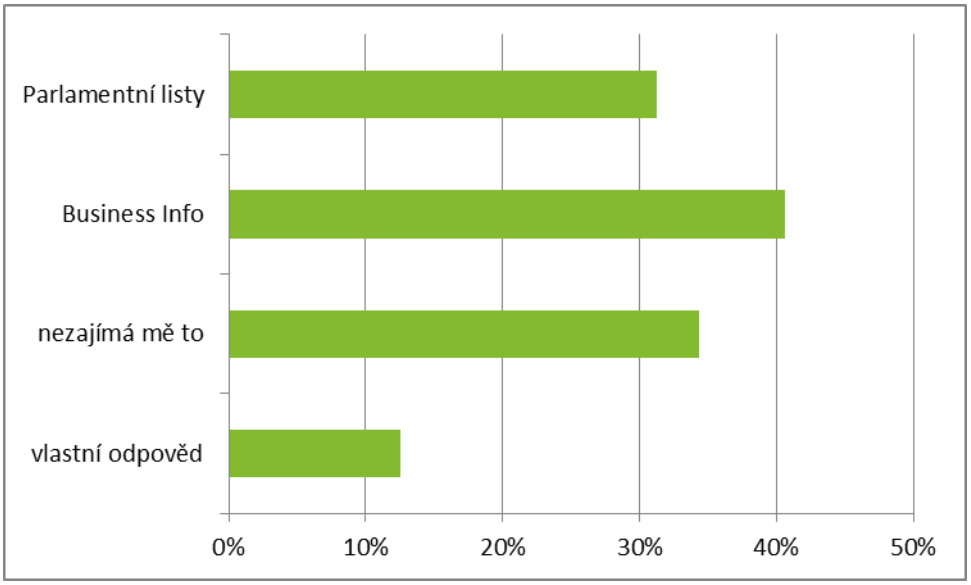
Celkově se nám podařilo získat 132 vyplněných dotazníků, ale 4 jsme byli nuceni vyřadit, protože byli neúplné. Při vyhodnocování odpovědí první otázky musíme brát v potaz, že respondenti mohli odpovědět buď zaškrtnutím poslední možnosti v případě, že nemají o veřejnou správu vůbec žádný zájem anebo jednoznačným určením trojice ze zbývajících možností. Z toho tedy vychází, že celkem 44% dotazovaných, buď uvedené webové stránky vůbec nesledují, nebo čtou jiné zpravodajské weby. Ale k tomu se dostaneme ještě detailněji u následující otázky. Zbylých 66% respondentů vybralo tedy nějakou trojici možností. Proto, abychom tedy vybrali, ze kterých webových stránek budeme získávat zprávy do nově vznikající Android aplikace, nám postačí znát četnosti, kolikrát byla trojice tvořena danou stránkou. Jak je z grafu znázorněném na obrázku č. 2 patrné, aplikace bude stahovat informace z webových stránek *České Vlády* (Vláda ČR, 2009), *prezidenta ČR a Pražského hradu* (Pražský hrad, 2014) a *Poslanecké sněmovny* (Parlament České republiky, 1997). Možnost v podobě Senátu ČR jsme zavrhli kvůli jejímu jednoznačně nejmenšímu zastoupení ve všech vybraných trojicích.



Obr. 2 Četnost zaškrtnutých možností v dotazníku u otázky č. 1

Ve druhé a zároveň poslední otázce jsme zjišťovali, které další zpravodajské weby lidé preferují. K dispozici měli i možnost napsat svůj vlastní zdroj informací, kterou využilo 13% respondentů. Nejčastěji zmiňovali ostatní média jako např. Českou televizi, rozhlas nebo jiné internetové portály. Ostatní si vystačili s nabízenými možnostmi. Největší zájem zaznamenal web *Business Info*, který si vybralo 41%

všech dotazovaných. (BusinessInfo.cz, 1997) Následoval ho portál *Parlamentní listy* s 31% zastoupením. (ParlamentniListy.cz, 2009) Někteří respondenti dokonce vybrali obě dvě zmíněné možnosti. A nakonec většina lidí, kteří již v předchozí otázce projevili nezájem o veřejnou správu i zde odpověděli záporně, tedy, že zvolili třetí možnost „nezajímá mě to“.



Obr. 3 Četnost zaškrtnutých možností v dotazníku u otázky č. 2

2.3 Návrh marketingové strategie

S ohledem na již dostupné Android aplikace podobného zaměření, které jsou k dispozici na Google Play zcela zdarma, nemá smysl uvažovat o zpoplatnění. Funkce, které budeme poskytovat navíc oproti dříve zmíněným aplikacím, jsou zaměřeny spíše na menší skupinu lidí, což je i dobře viditelné z odpovědí v dotazníku. Téměř polovina dotázaných nejeví o tuto oblast veřejné správy žádný zájem a zbytek se o ni zajímá spíše jen okrajově. Takže možnost přehrávání aktuální videa z jednání Poslanecké sněmovny či zjistit si jeho program bude spíše cílit na lidi aktivně působící v politice, veřejné správě či podnikatele.

Android aplikace bude taktéž zvyšovat povědomí a upevňovat pozici na trhu soukromé společnosti, v níž jsem vykonával pracovní stáž. A to tím způsobem, že při spouštění bude zobrazeno logo a jedna část aplikace bude sloužit k představení jejích aktivit a služeb.

Výběr vhodného způsobu distribuce provedeme, až bude vývoj aplikace celý dokončen, ale kritéria si můžeme definovat již teď. Největší vliv budou mít jednoznačně náklady, které by vzhledem k nulové ceně měli být minimální nebo žádné. Distribuce by měla pokrýt region tvořený alespoň Českou republikou v ideálním případě doplněný i o Slovenskou republiku. Pro ostatní státy není aplikace dostatečně perspektivní kvůli jejímu zaměření.

3 Operační systém Android

Přestože jsme si již něco o tomto relativně novém operačním systému pro chytrá mobilní zařízení něco málo řekli v první kapitole, nyní se podíváme podrobněji na jeho jádro a možnosti, které nabízí při programovaní aplikací určených pro tuto platformu.

3.1 Základní informace

Android je rozsáhlý operační systém vytvořený společností Google, založený na open source platformě, tedy jedná se o počítačový software s otevřeným zdrojovým kódem. Slova „otevřený kód" zde reprezentují snadnou dostupnost, a to jak technickou, tak licenční. Jinak řečeno, uživatel může systém při splnění jistých podmínek využívat zadarmo a tato licenční politika mu také umožňuje přistoupit ke zdrojovým kódům, které následně využívá nebo upravuje podle svých potřeb. OS je založen na Linuxovém jádře 2.6 různých verzí, které zajišťuje zabezpečení systému jako celku, správu paměti, správu procesů, přístup k síti a ovladačům všech vnitřních senzorů a komponent. Jednotlivé aplikace k funkcím jádra nepřistupují přímo, ale prostřednictvím Android API. (Ujbányai, 2011)

3.2 Základní kameny aplikace

Když píšete aplikaci pro stolní počítač, jste „pánem své vlastní domény". Spustíte její hlavní okno a tolik oken jeho potomků – například dialogů -, kolik potřebujete. Z vašeho úhlu pohledu se jedná o váš vlastní svět, ve kterém sice využíváte vlastnosti operačního systému, ale příliš se nestaráte o další programy, které mohou být v počítači spuštěné současně s vaším. Pokud s nimi nějak interagujete, děláte to obvykle prostřednictvím nějakého API, jako je například Java Database Connectivity (JDBC) nebo na něm založený aplikační rámec, abyste mohli komunikovat s MySQL nebo jinými databázemi.

Systém Android pak využívá podobné koncepce, ale jinak zabalené a strukturované s ohledem na větší ochranu telefonů proti zhroucení systému. Aplikace pro systém Android sestávají z následujících hlavních komponentů (Murphy, 2011):

- Aktivity - stavebními kameny pro tvorbu uživatelského rozhraní jsou aktivity. Aktivity můžete chápat jako analogii oken či dialogů aplikace pro stolní počítač. Ačkoliv je možné, aby aktivity neměly uživatelské rozhraní, většina vašeho zdrojového kódu bez uživatelského rozhraní bude zabalená spíše ve formě dodavatelů obsahu nebo služeb.
- Dodavatele obsahu – dodavatele obsahu zajišťují úroveň abstrakce jakýchkoliv dat uložených v zařízení, která jsou přístupná více různým aplikacím. Vývojový model systému Android vás podporuje v tom, abyste zpřístupnili svá data i jiným aplikacím než pouze své vlastní. Dosáhnete toho právě vytvořením do-

davatele obsahu, který vám současně poskytuje úplnou kontrolu nad způsobem přístupu k vašim datům.

- Služby – aktivity a dodavatele obsahu jsou entity s krátkou životností a lze je kdykoliv vypnout. Služby jsou oproti tomu navržené tak, aby, pokud je to potřeba, pokračovaly ve své práci nezávisle na jakékoliv aktivitě. Službu můžete použít k detekci aktualizaci RSS zdroje nebo k přehrávání hudby, které pokračuje, dokonce i když už byla příslušná aktivita uzavřena.
- Záměry – záměry jsou systémové zprávy upozorňující aplikace na výskyt různých událostí, změnami hardwarové konfigurace počínaje (například vložení SD karty) přes události související s předchozími daty (například přijetí SMS zprávy) a událostmi aplikace konče (například její spuštění z hlavního menu zařízení). Na záměry pak můžete nejenom reagovat, ale můžete je využívat k detekci specifických situací (například vyvoláte takový a takový záměr, když se zařízení dostane do vzdálenosti 100 metrů od takové a takové lokace).

Systém Android nabízí množství vlastností a nástrojů, které vám při vývoji aplikací pomáhají:

- Úložiště – datové soubory, jejichž obsah se nebude nijak měnit (například ikony nebo pomocné soubory), můžete přibalit přímo k vaší aplikaci. Navíc také můžete ukořistit trochu volného místa v zařízení a uložit data do databází nebo souborů obsahujících uživatelem zadávaná či načítaná data, která vaše aplikace potřebuje.
- Síť – zařízení využívající operační systém Android jsou obecně vzato připravená pro připojení k Internetu prostřednictvím různých druhů komunikačních médií. Přístup k Internetu můžete využít na jakékoliv úrovni chcete, surovými Java sockety počínaje a zabudovaným widgetem webového prohlížeče založeným na jádře WebKit, který můžete použít ve vaší aplikaci, konče.
- Multimédia – zařízení využívající operační systém Android umí přehrávat a zaznamenávat audio-videozáznamy. Ačkoliv se konkrétní specifika těchto schopností liší zařízení od zařízení, můžete se zařízení zeptat, jakými schopnostmi pro přehrávání a záznam audio- a videozáznamů disponuje, a poté tyto schopnosti využít, jak uznáte za vhodné – ať už k přehrávání hudby, pořizování fotografií fotoaparátem nebo záznamu audiopoznámek prostřednictvím mikrofonu.
- Global positioning systém (GPS) – zařízení využívající operační systém Android mají často přístup k dodavatelům údajů o zeměpisné poloze, například ke službě GPS, které vám mohou poskytnout informace o tom, kde na zeměkouli se zařízení zrovna nachází. Vy pak můžete následně na základě těchto informací zobrazovat mapy nebo je využívat nějak jinak, například ke sledování pohybu zařízení v případě jeho odcizení.
- Telefonní služby – protože jsou zařízení využívající operační systém Android obvykle telefony, může váš software samozřejmě také provádět telefonická

volání, zasílat a přijímat SMS zprávy a provádět jakékoliv další operace, které od moderního telefonu očekáváte.

3.2.1 Android Manifest

V předešlé kapitole jsme si představili základní stavební kameny aplikace určené pro operační systém Android. Aby aplikace fungovala jak má, je třeba operačnímu systému sdělit, jaké komponenty jsou k dispozici. K tomuto účelu Android Manifest. Jedná se o XML soubor, uložený v kořenovém adresáři, který specifikuje parametry aplikace – název, verze, jednotlivé komponenty, požadovaná systémová práva a další požadavky pro samotný chod aplikace. Android Manifest by měl také obsahovat (Ujbányai, 2011):

- Jméno Java balíčku aplikace, který reprezentuje unikátní identifikátor pro danou aplikaci.
- Deklaraci použitých komponent v aplikaci – aktivit, služeb, poskytovatelů obsahu atd., včetně názvů tříd, které implementují všechny komponenty, pomocí kterých publikují své schopnosti. O těchto zmiňovaných deklaracích musí být systém Android informován, aby byl schopen rozhodnout, za jakých podmínek mohou být spuštěny.
- Deklaraci oprávnění aplikace, které následně určují přístup ke chráněné části API funkcí a interakcí s ostatními aplikacemi (např. přístup na internet, použití SD karty atd.)
- Deklaraci minimální úrovně Android API, kterou aplikace vyžaduje.
- Seznam knihoven, prostřednictvím kterých musí být aplikace propojena

3.2.2 Úrovně Android API

Pro vývoj android aplikací je užitečné pochopit systém úrovní Android API a jejich rolí, jež hrají důležitou úlohu v zajištění kompatibility mezi naprogramovanými aplikacemi a přístroji, do kterých mohou být instalovány.

API úroveň reprezentuje celé číslo, které jednoznačně identifikuje rámec revize API, jenž je poskytován prostřednictvím dané verze Android platformy. Platforma Android tedy poskytuje *rámec API*, který mohou aplikace využívat pro komunikaci se základním systémem OS Android. Rámec API se skládá z následujících komponent (Ujbányai, 2011):

- Sada balíčku a tříd
- Sada elementů a atributů pro deklaraci v souboru AndroidManifest.xml
- Sada elementů a atributů pro deklaraci a přístup k prostředkům (zdrojům)
- Soubor záměrů
- Sada oprávnění, která aplikace mohou požadovat

Aktualizace rámce API je navržena tak, aby novější API byli kompatibilní se staršími verzemi, tzn. většina změn v rámci API je aditivní (dochází k přidání nové

funkčnosti, popř. přepisu té stávající). Pro udržení kompatibility již existujících aplikací nedochází v rámci API k odstranění zastaralých částí.

Jak již bylo řečeno, API rámec poskytovaný platformou Android je identifikován číslem, které se nazývá *úroveň API*. Každá verze platformy Android podporuje přesně jednu úroveň API. První verze OS Android poskytla rozhraní úrovně API 1. U všech následujících verzí Android byla úroveň API, resp. Celé číslo, kterým je označována, zvýšeno o jedničku.

Tab. 1 Přehled verzí platformy Android a poskytovaných úrovní API

Verze platformy Android	Poskytovaná úroveň API
Android 4.0, 4.0.1, 4.0.2	14
Android 3.2	13
Android 3.1.x	12
Android 3.0.x	11
Android 2.3.3 / 2.3.4	10
Android 2.3 / 2.3.1 / 2.3.2	9
Android 2.2.x	8
Android 2.1.x	7
Android 2.0.1	6
Android 2.0	5
Android 1.6	4
Android 1.5	3
Android 1.1	2
Android 1.0	1

Zdroj: Ujbányai, 2011 (s. 48).

3.3 Vývojové prostředí

Předpokladem tvorby aplikací pro Android je připravené prostředí pro samotný vývoj a ladění aplikací označované anglickou zkratkou *IDE (Integrated Development Environment)*. Je tedy potřeba mít nainstalované vývojové prostředí, sadu nástrojů potřebnou k programování (Android SDK) a ADT plugin. V neposlední řadě je potřeba mít nastaveno virtuální zařízení, na kterém budeme aplikace spouštět, popřípadě ladit, pokud nedisponujeme fyzickými zařízeními s OS Android.

Pokud se podíváme do jakékoliv dříve zmíněné literatury, snad ve všech je vysvětlován postup tvorby aplikace za pomoci vývojového prostředí nazývaného Eclipse. Což lze jednoduše zdůvodnit skutečností, že společnost Google toto vývojové prostředí donedávna doporučovala a podporovala. V porovnání s ostatními možnosti v podobě NetBeans či IntelliJ IDEA bylo Eclipse nejlépe zdokumentované a proto jej většina mobilních vývojářů používala. NetBeans IDE vlastněné společností Oracle bylo a stále je díky své modulární architektuře i přes své primární určení poměrně hodně používáno webovými vývojáři, ale málokdo jej používá pro

vývoj Android aplikací. Trochu v pozadí bylo donedávna prostředí IntelliJ IDEA, což se změnilo s příchodem beta verze Android Studia postaveném na jeho základech. Konkrétně na volně dostupné verzi Community Edition, která je na rozdíl od Ultimate Edition zdarma. Nyní vyšla první finální verze a oproti ostatním vývojovým prostředím je co nejvíce zaměřena na mobilní vývojáře. Ulehčuje jim práci mnoha různými způsoby, tak si představíme alespoň některé. Zdaleka největší výhodu přináší zabudovaní automatizačních nástrojů Gradle a Maven, díky nimž je integrace doplňkových knihoven otázkou dvou kliknutí stejně jako generování různých verzí hotového instalačního balíčku dané aplikace. Dále umožňuje snadný přístup k ukázkám zdrojového kódu, vylepšení grafický XML editor, integraci se systémem pro správu verzí Git a také nově vzniklou platformou Google Cloud Platform pro tvorbu webového API. Díky tomu máme rozhodování dosti ulehčené a je zřejmé, že programovat tedy budeme v prostředí Android Studia.

Pro spouštění aplikací jsou samozřejmě nejlepší reálná zařízení a i my budeme po většinu času tuto variantu používat, ale pro testování a ověření funkčnosti i na odlišných zařízeních než máme k dispozici, je vhodné provést jejich virtualizaci. To sice sada Android SDK umožňuje, ale její řešení není zrovna svižné. Použitelné se stává snad jen na počítačích s nejnovějšími procesory Intel. Mnohem efektivnější způsob nabízí *Genymotion*, který virtualizaci provádí s pomocí oblíbeného programu VirtualBox a jeho rychlost je v porovnání se standardním Android Emulátorem naprosto ohromující. (Genymotion, 2014)

3.4 Komunikace s API

V systému Android nejsou zabudovaná klientská API protokolů SOAP nebo XML-RPC. Avšak jeho nedílnou součástí je knihovna Apache HttpComponents. Nad touto knihovnou pak můžete buď umístit vrstvu protokolů SOAP/XML-RPC, anebo ji používat „přímo“ k přístupu k webovým službám založeným na architektuře REST. Pro potřeby této knihy považujeme webové služby založené na architektuře REST za jednoduché http požadavky na běžná URL s plnou škálou http metod (sloves), které vrací formátovaná data (např. ve formátu XML, JSON atd.). (Murphy, 2011)

Knihovna HttpClient, která je součástí knihovny HttpComponents, obsluhuje všechny http požadavky za vás. Prvním krokem při práci s knihovnou HttpClient je vytvoření objektu typu *httpClient*. Protože se jedná o rozhraní, bude muset ve skutečnosti vytvořit instanci nějaké implementace tohoto rozhraní, např. třídy *DefaultHttpClient*.

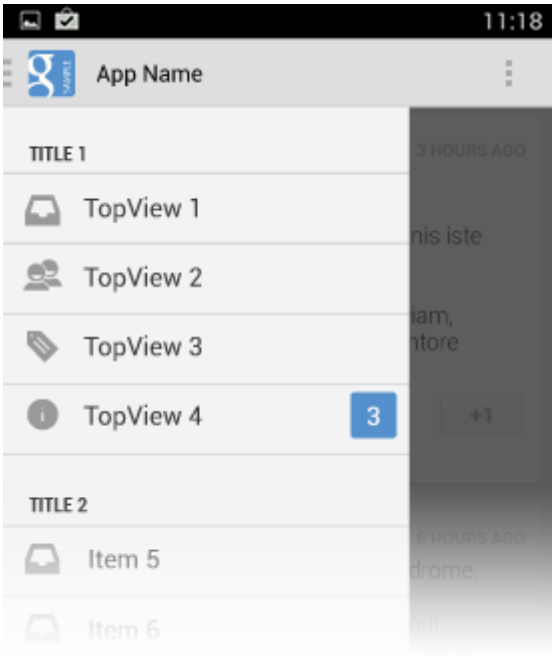
Požadavky se zabalí do instancí třídy *HttpRequest* s různými implementacemi pro jednotlivá HTTP slovesa (např. *HttpGet* pro http požadavek *GET*). Vy vytvoříte instanci třídy *httpRequest* s příslušnou implementací, vložíte do ní cílové URL a další konfigurační údaje (např. hodnoty formulářů, když provádíte http požadavek *POST* prostřednictvím instance třídy *HttpPost*) a poté předáte metodu klientu, aby http požadavek provedl prostřednictvím metody *execute()*.

Co se stane v tuto chvíli, může být velmi jednoduché nebo také velmi složité – to záleží jenom na vás. Vrátit můžete objekt typu *HttpResponse* s kódem odpovědi

(např. 200, pokud vše proběhlo tak, jak mělo), http hlavičky apod. Nebo můžete použít některou z verzí metody *execute()*, která přijímá jako parametr objekt typu *ResponseHandler<String>* výsledkem bude, že metoda *execute()* vrátí pouze řetězcovou reprezentaci těla odpovědi. Tento postup se však v praxi příliš nedoporučuje, protože byste opravdu měli kontrolovat vaše kódy http odpovědí a zjišťovat, zda neobsahují nějaké chyby. Avšak v případě triviálních aplikací, například takových, které se uvádějí v knihách jako příklady, tento postup funguje docela dobře.

3.5 Současné vývojové metody

- Navigace v drtivé většině aplikací je v současné době řešena návrhovým vzorem Navigation drawer. Ten se dá popsat jako panel, který přechází v levý okraj obrazovky a zobrazuje hlavní menu dané aplikace. Uživatel ho může zobrazit gestem v podobě přejetí prstem od levého okraje displeje směrem doprostřed nebo kliknutím na ikonu umístěnou v tzv. ActionBaru.



Obr. 4 Návrhový vzor Navigation Drawer
Zdroj: Navigation Drawer, 2011.

- V Androidu 3.0 byly poprvé představeny fragmenty a to především pro podporu více dynamického a flexibilního designu uživatelského rozhraní na velkých obrazovkách, které tablety nabízejí. Vzhledem k jejich velikosti je zde mnohem větší prostor pro kombinování a výměnu komponent uživatelského rozhraní. Fragmenty umožní takový design vytvořit bez nutnosti spravovat komplexní změny v celkové hierarchii komponent. Např. aplikace ke sledování aktualit může použít jeden fragment k zobrazení seznamu článků na levé stra-

ně a druhý k zobrazení samotného článku na opačné straně – oba fragmenty náleží do té samé aktivity a každý má svůj vlastní životní cyklus.

Obr. 5 Návrhový vzor Master/detail na tabletu a na chytrém telefonu
Zdroj: Building a Flexible UI, 2011.

4 Návrh a implementace aplikace

Aplikace vycházejí v mnoha variantách, protože musí splňovat různé potřeby. Můžeme si to ukázat na několika příkladech: aplikace jako je Kalkulačka nebo Fotoaparát tvoří jediná aktivita, která poskytuje všechny potřebné funkce na jediné obrazovce. Takové aplikace mají jeden hlavní účel, který spolehlivě plní a uživatel od nich ani nic víc nepotřebuje. Naproti tomu aplikace Telefon umožňuje přecházet mezi více různými aktivitami bez podrobnější navigace. Podobně jsou na tom Gmail nebo Google Play, které navíc kombinují širokou řadu obrazovek s detailnější navigací. A právě takovým typem aplikace bude i ta nově vznikající.

Proto budeme v průběhu navrhování naší aplikace dodržovat nejen doporučení v dokumentaci, ale použijeme i některé principy stejně jako vývojáři Gmailu. Jejich aplikace má v současnosti více než jednu miliardu spokojených uživatelů. To je samo osobě výbornou vizitkou a důkazem správně navrženého uživatelského rozhraní. Dalším vynikajícím příkladem, u kterého se můžeme při návrhu inspirovat je open-source aplikace *Google I/O 2014* sloužící jako ukázka použití nejnovějších postupů. (Google I/O Android App, 2014) Společnost Google zaštiťující její vývoj ji aktualizuje každý rok před začátkem stejnojmenné konference. Tato konference se pořádá pravidelně již od roku 2008 a bývá zde představena nová verze operačního systému Android přinášející řadu vylepšení zlepšující uživatelský zážitek.

4.1 Požadované vlastnosti

Vezmeme-li v potaz požadavky vyřčené při definování cíle této práce, tak by aplikace měla splňovat následující kritéria:

- Podpora starších verzí Android platformy – velké množství zařízení stále pohání Android 2.2 Froyo a 2.3 Gingerbread. Nejčastější příčinou je ukončená podpora od výrobce, který již nevydal aktualizaci pro tato zařízení. V současné době se již více projevuje trend, kdy je většina aplikací vyvíjena pro Android 4 Ice Cream Sandwich a novější. Starší verze Androidu nemají takové funkce a přednosti jako ty současné. Přesto existuje způsob, jak alespoň některé novinky lze používat i na těchto zastaralých verzích. Oním způsobem je knihovna s názvem *Support Library*. (Support Library, 2011) Díky ní docílíme stejného vzhledu aplikace, lepšího výkonu a oslovíme více uživatelů.
- Optimalizace GUI pro tablety – i když velikost chytrých telefonů stále roste, pořád platí, že tablety mají mnohem větší zobrazovací plochu, kterou by byla škoda nevyužít. Řešením může být návrhový vzor Master/detail, který se dokáže přizpůsobit jakékoliv uhlopříčce displeje.
- Podpora přehrávání videa na různých typech procesorů – bohužel Android nativně podporuje jen vybrané formáty médií, a pokud je video v podobě online streamu stejně jako v našem případě, je situace o to složitější. Porozhlédneme-li se po open-source přehrávačích multimédií, jež by vylo možné impor-

tovat do naší aplikace ve formě knihovny, moc toho na výběr nemáme. *ExoPlayer*, nový přehrávač médií od společnosti Google, bohužel jen nepatrně rozšiřuje možnosti v porovnání se systémovým přehrávačem a pro jeho funkčnost na starších zařízeních by bylo potřeba vyvinout vlastní způsob dekódování videa. (ExoPlayer, 2014) Kvůli časové náročnosti tedy tuto možnost zamítneme stejně tak jako knihovnu *FFMPEG*. Ta pro své použití vyžaduje detailní znalosti práce s médii a používá se spíše při vývoji samotného multimediálního přehrávače. Například právě projekty jako *VLC* (VLC media player, 2014) či *Vitamio* (Vitamio for Android, 2014) ji používají a proto bychom je mohli zařadit mezi zvažované možnosti. Oba mají podobné vlastnosti: jednoduchý způsob implementace a širokou podporu nejrůznějších formátů videa. Ale ve prospěch Vitamia hovoří snadný import do našeho projektu za pomoci pluginu Gradle. U VLC bychom museli použít rozšíření Android NDK podporující kód napsaný v programovacím jazyce C nebo C++ a pro každý typ architektury procesoru si zkompilovat speciální soubory.

- Univerzální způsob parsování RSS – RSS je rodina XML formátů určených k syndikaci obsahu. V dnešní době se používají 2 hlavní specifikace určující výslednou podobu souboru, a to jsou RSS 2.0 společně s Atomem 1.0. Vzájemně se odlišují v používaných elementech a jejich struktuře. Rozdíl není ani tolik markantní, ale pokud bychom ho opominuli, zbytečně bychom si zadělali na problémy a data by nebyla kompletní.
- Webová komunikace s API – ne všechny webové stránky mají vlastní RSS kanál. Nebo i přestože ho mají, neobsahuje požadované informace a ty jsou tak dostupné jen skrze webový prohlížeč. Nabízí se tedy otázka: jakým způsobem je získat do naší aplikace? Elegantním a zároveň sofistikovaným řešením může být vlastní webové API postavené na architektuře REST. To nám umožní jednotný a snadný přístup ke zdrojům. Zdroje budou v našem případě ve formátu JSON, který nám usnadní ukládání dat v zařízení. Komunikace mezi samotnou aplikací a API bude v podobě běžného požadavku na stránku, tedy metody *GET* protokolu HTTP.

4.2 Základní struktura

Ale přejděme již k samotné struktuře aplikace. Jak víme, základními stavebními prvky jsou aktivity a fragmenty. Použití fragmentů je neoddiskutovatelné, ale nabízí se jiná otázka: z kolika aktivit se bude skládat naše aplikace? Je lepší použít jednu nebo více aktivit? Řešení v podobě jedné aktivity se používá v případech, kdy napříč celou aplikací zůstává layout neměnný, např. každou sekci tvoří jediný fragment přes celou obrazovku. Toto řešení je také vhodné, pokud má celá aplikace v jakékoliv situaci reagovat identicky. Třeba, aby se při změně orientace vždy aktivita restartovala (v životním cyklu aktivity tzn. volání metod *onDestroy()* a *onCreate()*). Druhé řešení s více aktivitami se používá v případech, kdy je layout napříč aplikací odlišný (různé části aplikace jsou tvořeny různým počtem fragmentů) a aktivity mohou na změnu orientace a podobné situace reagovat odlišně.

V naší situaci použijeme druhou variantu, neboť ne všechny aktivity budou pracovat stejným způsobem. Jedna skupina se bude opětovně spouštět při změně orientace obrazovky. Do ní bude patřit drtivá většina aktivit, neboť návrhový vzor Master/detail funguje spolehlivě jen díky tomuto principu. Druhá skupina je tvořena aktivitami obstarávající práci s videem a s vlákny běžícími na pozadí. Začněme tedy s implementací, spustíme editor, založíme nový projekt a můžeme se pustit do tvorby základního menu.

4.2.1 Menu

Při rozhodování jakou podobu menu zvolit se nebudeme odchylovat od současného trendu a použijeme návrhový vzor Navigation Drawer. Díky Android Studiu je jeho implementace opravdu snadná. Buď již při vytváření nového projektu můžeme zatrhnout možnost vložení aktivity v této podobě nebo pouhým kliknutím pravým tlačítkem myši na složku *java* a následnou volbou *Navigation Drawer Activity*. Obě tyto operace nám vytvoří základní šablonu společnou pro všechny hlavní aktivity, kterou si teď přizpůsobíme našim potřebám.

Již teď víme, že aplikace bude obsahovat několik různých tematických sekcí. Aby se uživatel mezi nimi lépe orientoval, odlišíme je od sebe hlavičkou s nápisem odlišné barvy a kapitálkami. Takového vzhledu dosáhneme úpravou adaptéru obstarávající zobrazování jednotlivých položek menu.

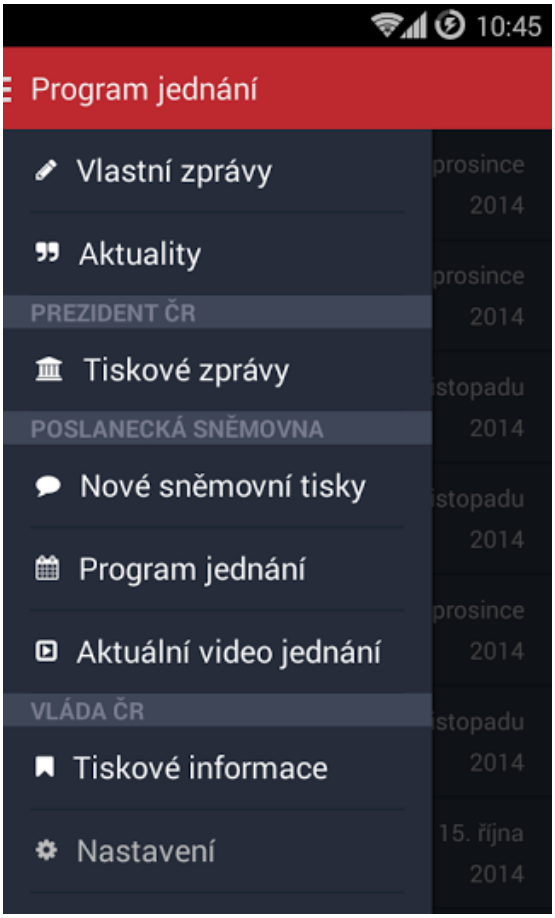
```
class NavDrawerAdapter extends BaseAdapter {
      LayoutInflater mInflater;
      List<Integer> titles = new ArrayList<Integer>(Arrays.asList(1,
      3));
      String[] icons = getResources().getStringArray(R.array.icons);
      String[] items = new String[]{
            getString(R.string.title_section1),
            getString(R.string.title_section2),
            getString(R.string.title_section3),
            getString(R.string.title_section4),
            getString(R.string.title_section5)
      };
      String[] headers = new String[]{
            getString(R.string.header_section1).toUpperCase(),
            getString(R.string.header_section2).toUpperCase()
      };
final class ViewHolder {
            FontAwesomeText icon;
            TextView textHeader;
            TextView textItem;
        }
```

Princip úpravy je snad zřejmý, do layoutu každé položky v menu přidáme navíc prvek *TextView* s jediným účelem, zobrazení textu hlavičky. Přidaný prvek bude viditelný pouze v položkách na vybraných pozicích. Pozice, kde se budou hlavičky zobrazovat, tedy obsahuje seznam *titles* a pole *headers* obsahuje jejich text. Algoritmus rozhodování, zda hlavičku zobrazit nebo ne je uveden níže. K němu je jed-

ním dechem potřeba dodat, že se nachází uvnitř metody *getView*, jenž je zároveň součástí třídy *NavDrawerAdapter* a také, že tento adaptér uplatňuje návrhový vzor ViewHolder pro rychlejší vykreslování a lepší výkon.

```
if (titles.contains(i)) {
holder.textHeader.setVisibility(View.VISIBLE);
      switch (i) {
            case 1:
                  holder.textHeader.setText(headers[0]);
                  break;
            case 3:
                  holder.textHeader.setText(headers[1]);
            }
      } else {
            holder.textHeader.setVisibility(View.GONE);
      }
```

Výsledek úpravy si můžeme prohlédnout na obrázku č. 6, kde lze spatřit i použité ikony. Ty nejsou implementovány standardním způsobem, ale pomocí knihovny *Android-Bootstrap*. (Android-Bootstrap, 2014) Jejím hlavním účelem je sice zaručení stejného vzhledu tlačítek jaké používá populární webový framework Bootstrap, ale my ji použili převážně z jiného důvodu. Dokáže totiž správně vykreslovat specifické písmo se jménem *Awesome Font*, jenž obsahuje kromě obyčejných znaků dokonce i ikony. A stejně jako u kteréhokoliv jiného písma můžeme měnit jeho barvu, buď přímo v XML souboru definující *layout* anebo ve zdrojovém kódu. Taková vlastnost dokáže ušetřit velkou spoustu času. Vždyť každý ví, kolik dá práce a námahy si vyrobit vlastní ikony ve všech možných rozlišeních a požadované barvě. Velký výběr nenabízí ani základní balíček ikon, jenž lze získat na webových stránkách dokumentace. V tomto ohledu je tedy Android-Bootstrap bezkonkurenční.



Obr. 6 Hlavní menu celé aplikace

Nabízí široký výběr ikon ve všech dostupných rozlišeních, jakékoliv barvě, má několik předdefinovaných animací a s jeho přičiněním je věc jako sjednocení designu celé aplikace neuvěřitelně lehkou záležitostí. Jeho začlenění provedeme následovně, před prvek *TextView* vložíme *FontAwesomeText*. A to celé bude součástí *LinearLayoutu*, u kterého nastavíme atribut *orientation* na hodnotu *horizontal*. Tak bude zajištěno srovnání jednotlivých prvků pěkně do řádku.

```
<LinearLayout
android:id="@+id/item"
      android:layout_width="match_parent"
      android:layout_height="?attr/actionBarSize"
      android:gravity="center_vertical"
      android:orientation="horizontal"
      android:paddingBottom="8dp"
      android:paddingLeft="@dimen/activity_vertical_margin"
      android:paddingRight="@dimen/activity_vertical_margin"
      android:paddingTop="8dp">

<com.beardedhen.androidbootstrap.FontAwesomeText
            android:id="@+id/icon"
            android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:textAppearance=
            "?android:attr/textAppearanceMedium" />

        <TextView
            android:id="@+id/item_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="8dp"
            android:textAppearance=
            "?android:attr/textAppearanceMedium"
            android:textColor="@android:color/primary_text_dark" />
</LinearLayout>
```

Menu aplikace tedy můžeme považovat za hotové a teď jej potřebujeme dostat nějakým způsobem do všech hlavních aktivit, v nichž se bude vyskytovat. Z tohoto důvodu si vytvoříme rodičovskou třídu *BaseActivity*, jenž bude základním stavebním kamenem celé aplikace. Stejný princip používá i aplikace Google I/O 2014 a žádné jiné řešení snad ani není kromě neefektivního znovu-opisování kódu do každé aktivity zvlášť. Abychom zajistili zpětnou kompatibilitu až do úrovně API 8, použijeme k tomu určený balíček knihoven Support Library. Z něj budeme konkrétně využívat knihovny v4-support a v7-appcompat. První z nich poskytuje možnost používat fragmenty už od verze API 4, přestože fragmenty jako takové byly představeny až mnohem později. A druhá nám zaručí podobný vzhled aplikace díky schopnosti importovat námi aplikovaný návrhový vzor Navigation Drawer i ActionBar až na API verze 7. Přitom bychom místo ní mohli použít i alternativní variantu *ActionBarSherlock*, která dokonce byla donedávna mnohem vhodnější. (ActionBarSherlock, 2014) Ale v současnosti jsou na tom obě velice podobně, takže ve prospěch v7-appcompat hovoří skutečnost, že používá totožnou implementaci stejně jako operační systém za všech okolností. Tím je zaručen stejný uživatelský zážitek na jakémkoliv zařízení a nemusíme se obávat nečekaných bugů, které přináší vlastní implementace jako v případě knihovny ActionBarSherlock.

Takže v7-appcompat importujeme do *BaseActivity* pouhým děděním, rodičem musí být třída *ActionBarActivity*. Tímto se tedy změní práce se samotným ActionBarem, takže se od této chvíle se bude volat metodou *getSupportActionBar* namísto původního *getActionBar*. Aktivita bude také implementovat rozhraní definované fragmentem *NavigationDrawerFragment* a to *NavigationDrawerCallbacks*. Rozhraní obsahuje jedinou metodu *onNavigationDrawerItemSelected* s parametrem *position* dožadující se nějaké reakce při stisku určité položky v menu.

```
@Override
public void onNavigationDrawerItemSelected(final int position) {
if (position < 3) {
            mHandler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    goToNavDrawerItem(position);
                }
```

```
            }, getResources().
            getInteger(android.R.integer.config_shortAnimTime));
            if (fade) {
                final View mainContent = findViewById(R.id.container);
                if (mainContent != null) {
                    mainContent.startAnimation(animation);
                }
            }
        } else {
            goToNavDrawerItem(position);
        }
    }
```

Jak je ze zdrojového kódu patrné, v našem případě metoda reaguje na stisk tlačítka s drobným zpožděním okolo 150ms. A stejně jako v předchozím případě je tento princip na zlepšení uživatelského zážitku převzat z aplikace Google I/O 2014. Snažíme se tím zamaskovat ne příliš plynulý přechod mezi aktivitami. Běžně to totiž vypadá tak, že po stisku tlačítka v menu se současně začne spouštět nová aktivita a zároveň zavírat menu. To je však příliš výpočetně náročné na procesor a výsledkem je opravdu pomalé spouštění nové aktivity doprovázené trhaným obrazem. Jedinou výjimkou jsou aktivity jednoduššího charakteru, mezi které patří například aktivity pracující s úložištěm SharedPeferences. V naší aplikaci jedna taková aktivita je, nazývá se *SettingsActivity* a v menu nachází se na poslední pozici. Na ni tedy budeme reagovat okamžitě, u ostatních položek reakci chvíli zdržíme a dostupný výkon procesoru využijeme k vykreslení animace zavření menu a schování obsahu. Kvůli zpětné kompatibilitě voláme metodu *startAnimation()* namísto novější *animate()*.

Další nešvar doprovázející námi zvolenou strukturu aplikace z více aktivit se projevuje v situacích, kdy v menu vybereme aktivitu shodnou s tou, v níž se aktuálně nacházíme. V tom případě se aktivita spustí znovu od začátku a uživatele to bezpochyby nenechá chladným. Přitom ošetření takového problému není vůbec složité. Stačí do zdrojového kódu přidat podmínku, zda je aktuální aktivita instancí té nově zvolené.

```
private void goToNavDrawerItem(int i) {
Intent intent;
      switch (i) {
            case 0:
                if (!(this instanceof NewsActivity)) {
                    intent = new Intent(this, NewsActivity.class);
                    intent.putExtra(TAG_ANIMATE, true);
                    startActivity(intent);
                    overridePendingTransition(0,
                            0);
                    finish();
                    fade = true;
                }
                break; …
```

4.2.2 Ukládání dat

Předtím, než postoupíme dále, si ještě zvolíme typ úložiště, kde budeme ukládat potřebná data a dále knihovnu *Calligraphy* umožňující nám použít libovolný font v kterékoliv části aplikace. (Calligraphy, 2014) Samotný Android totiž poskytuje pouze jeden s názvem Roboto, což je dosti omezující. Jedinou možnost jakou nám platforma dovoluje jej změnit je přímo ve zdrojovém kódu, a to není příliš praktické řešení. Calligraphy nám otevírá zcela nové možnosti v této oblasti. Stačí si nadefinovat vlastní atribut *fontPath*, přidat písma do složky *assets/fonts*, inicializovat v třídě typu *Application* a samotné aktivitě, poté již můžeme při návrhu layoutu uvádět zmíněný parametr a jeho atributem bude samotný název písma.

Při rozhodování jakým způsobem ukládat data máme k dispozici hned několik možností. Než si je ale všechny představíme, zamysleme se chvilku nad strukturou našich dat. Při definování požadavků, jenž má naše aplikace splňovat, jsme konstatovali fakt, že webové API nám bude poskytovat data v souboru formátu JSON. To protože Android z něj dokáže získat potřebné informace s opravdovou lehkostí a i díky tomu je tento způsob hojně používaný v ostatních mobilních aplikacích. Druhým typem dat budou strukturované textové řetězce získané z jednotlivých RSS kanálů specifickou metodou, která pro nás v tuto chvíli není vůbec podstatná. Strukturu našich dat tedy známe, nyní přejděme k výběru nejvhodnější varianty jejich uložení.

Pakliže bychom ukládali pouze několik málo hodnot a k jejich odlišení by nám stačilo pár unikátních názvů, ideální volbou by byli SharedPreferences. Tento způsob se hodí v případech, kdy si potřebujeme uchovat např. preferovaný tón vyzvánění nebo číslo kolikrát byla aplikace spuštěna a my mohli při prvním spuštění ukázat uživateli návod či instrukce jak ovládat danou aplikaci. Což se pro žádnou strukturu našich dat nehodí. Dalším možností na řadě je tedy interní a externí úložiště telefonu nebo tabletu. Rozdíl mezi nimi je v přístupnosti obsahu a také skutečnosti, že ne všechny zařízení mají externí úložiště neboli jinak řečeno paměťovou kartu. K souborům uloženým v interním úložišti má přístup pouze samotná aplikace a nikdo jiný. Kdežto externí úložiště dovoluje komukoliv číst, mazat nebo upravovat uchovávaný obsah. Jak je zřejmé úložiště tedy využijeme k uskladnění JSON souboru, otázkou ale zůstává, zda interní či externí.

Tuto volbu necháme na samotném operačním systému, protože naše aplikace nebude pracovat s žádnými citlivými údaji, jenž by byli potřeba nějakým způsobem chránit. Pokud bude na konkrétním zařízení dostupné externí úložiště, aplikace se nainstaluje právě na něj. V opačném případě bude instalace provedena do interní paměti. Tento způsob tedy vychází vstříc požadavkům dnešních uživatelů, kteří se snaží šetřit místo v paměti zařízení. Proto dopíšeme na začátek souboru AndroidManifest.xml atribut *installLocation* s parametrem *auto*.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="eu.imakers.asistent"
    android:installLocation="auto">
```

Pravděpodobně nejčastějším řešením ukládání dat na Androidu je integrovaná SQLite databáze, která má své zvláštnosti. Asi největším specifikem oproti ostatním je skutečnost, že nemá typovou kontrolu. Takže je možné do ní vkládat hodnoty odlišného typu, než jsme si stanovili bez jakéhokoliv upozornění. Její primární určení je skladovat a poskytovat strukturovaná data. Ale připravit si databázi, než budeme moci používat, je samo o sobě dost obtížné a zdlouhavé, zvláště ve spojení s ContentProvidery. Do češtiny se dá tento výraz přeložit jako „dodavatelé obsahu" a přesně to dělají, poskytují obsah databáze díky klasickým SQL dotazům a zjednodušují nám přístup k databázi. Ovšem jak jsme již řekli, jejich vytvoření je tím náročnější, čím je složitější model samotné databáze. My databázi i společně s ContentProviderem v naší aplikaci použijeme pro ukládání strukturovaných textových řetězců. Jenže my si její přípravu usnadníme za pomoci knihovny postavené na architektuře ORM. Nazývá se *ActiveAndroid* (ActiveAndroid, 2014) a dokáže opravdu radikálním způsobem zrychlit proces přípravy, protože dokáže sama spravovat databázi a na nás v podstatě zbyde jen samotná tvorba modelu. Podobných knihoven existuje ještě několik např. *greenDAO* (greenDao, 2014) či *Sugar ORM* (Sugar ORM, 2014), ale ActiveAndroid zajišťuje funkčnost i na starších verzích API a podle mého názoru je ze všech nejvíce intuitivní. Pojďme si jej tedy integrovat do naší aplikace.

Ještě než se pustíme do tvorby modelu databáze, je nutné provést několik specifických kroků. Takže si vytvoříme si novou třídu *BaseApplication* a v ní inicializujeme jak knihovnu ActiveAndroid, tak i dříve zmíněnou Calligraphy. Zároveň si dáme pozor, aby rodičovská třída byla z balíčku *com.orm* a ne standardního *android.app*. Výsledek by měl vypadat přibližně takto.

```
public class BaseApplication extends Application {

      @Override
      public void onCreate() {
        super.onCreate();
        ActiveAndroid.initialize(this);
        CalligraphyConfig.initDefault(R.attr.fontPath);
    }}
```

Následující krok spočívá v drobné modifikaci souboru AndroidManifest.xml. Do části *application* vepíšeme atribut *name* a jako parametr uvedeme nově vytvořenou *BaseActivity*. Ani nemusíme vypisovat celý název balíčku, ze kterého pochází, můžeme si ho zkrátit tečkou. Přidáme prvky označené tagem *meta-data* a v nich definujeme název databáze dohromady s třídami specifikující celkový model. Každá jednotlivá třída představuje právě jednu tabulku v databázi. V našem případě si tedy vystačíme s pouhými dvěma tabulkami: items a sources, které jsou popsány v třídách *Item* a *Source*. Nakonec přidáme prvek *provider*, kde musí být u atributu *name* uvedeno přesně to, co požaduje knihovna ActiveAndroid.

```
<application
android:name=".BaseApplication"
      android:allowBackup="true"
```

```
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme">
  <meta-data
           android:name="AA_DB_NAME"
           android:value="AsistentPoslance.db" />
       <meta-data
           android:name="AA_MODELS"
           android:value=".model.Item, .model.Source" />
       <provider
           android:name="com.activeandroid.content.ContentProvider"
           android:authorities="eu.imakers.asistent"
           android:exported="false" />
```

Konečně se tedy můžeme vrhnout na model naší databáze. V předchozích krocích jsme se zmínili o třídách *Item* a *Source*, které by měli být základní kostrou našeho modelu. Obě dvě jsou potomky třídy Model z balíčku *com/orm* a díky anotacím knihovny ActiveAndroid z nich budou vygenerovány tabulky v naší SQLite databázi. Třída *Item* představuje jednotlivé zprávy získané z různých RSS kanálů. Zahrnuje stejné atributy, jaké používá technologie RSS a její specifikace RSS 2.0 a Atom 1.0 tzn. *title*, *description*, *link*, *date*. Z jednotlivých názvů je snad patrné, co je jejich obsahem, ale pro jistotu si to řekneme. *Title* určuje nadpis nebo titulek, *description* je krátký popisek obsažené zprávy, *link* odkaz na webovou stránku, kde se zpráva nachází v kompletní podobě a nakonec *date* označuje datum jejího publikování. Posledním atributem, který je taktéž součástí třídy *Item*, je *source*. Ten pojímá cizí klíč druhé tabulky, která je definována v třídě *Source*. Vztah mezi těmito dvěma entitami je typu 1:N, jinak řečeno jedna entita třídy *Source* může poskytovat libovolný počet entit typu *Item*.

Třída *Source* obsahuje pár atributů, mezi které patří *name* a *url*. První z nich obsahuje název samotného feedu neboli RSS kanálu a ten druhý adresu, na které se konkrétní feed nachází. Dáme-li tohle všechno dohromady, vznikne nám výsledný model databáze zobrazený v podobě ERD diagramu nacházející se v příloze B. Z něj by to mělo být snadněji pochopitelné.

A pro kompletní dokončení ukázky, o kolik je jednodušší proces přípravy databáze s knihovnou ActiveAndroid, si prohlédneme i implementaci třídy *Item*. Z kódu je jasně patrné, že proměnné je potřeba opatřit speciálními anotacemi, aby z nich knihovna mohla získat všechny podstatné informace. Důležité je upozornit na anotaci id a metodu *fetchResultCursor()*, jenž knihovna požaduje při použití ContentProvideru. Názorně jde vidět definování vztahu mezi entitami.

```
@Table(name = "items", id = BaseColumns._ID)
public class Item extends Model {

    public static final String NAME_TITLE = "title";
    public static final String NAME_DESCR = "description";
    public static final String NAME_LINK = "link";
    public static final String NAME_DATE = "date";
    public static final String NAME_SOURCE = "source";
```

```
@Column(name = NAME_TITLE)
private String title;
@Column(name = NAME_DESCR)
private String description;
@Column(name = NAME_LINK)
private String link;
@Column(name = NAME_DATE)
private Date date;
@Column(name = NAME_SOURCE, onUpdate = Column.
  ForeignKeyAction.CASCADE, onDelete = Column.
  ForeignKeyAction.CASCADE)
private Source source;

public Item() {
    super();
}

public Item(String title, String description, String link, Date
date, Source source) {
    super();
    this.title = title;
    this.description = description;
    this.link = link;
    this.date = date;
    this.source = source;
}

public static Cursor fetchResultCursor() {
    String tableName = Cache.
        getTableInfo(Item.class).getTableName();
    // Query all items without any conditions
    String resultRecords = new Select(tableName + ".*, "
        + tableName + ".Id as _id").
            from(Item.class).toSql();
    // Execute query on the underlying ActiveAndroid SQLite
       database
    return Cache.openDatabase().rawQuery(resultRecords, null);
}
```

4.2.3 Uživatelské rozhraní

Nyní přichází na řadu otázka, jakým způsobem získáme potřebná data, které bychom mohli do již připravené databáze ukládat. Jak víme z předchozích odstavců, data budou pocházet konkrétních RSS kanálů jednoznačně určenými přesnou adresou tedy URL. Vytvoříme si tedy univerzální sekci aplikace, v jejímž rámci si uživatel bude moci spravovat své vlastní informační kanály. Později její metody využijeme i v ostatních částech aplikace, neboť budou fungovat na společném principu.

Celá sekce bude implementována návrhovým vzorem Master/detail, se kterým jsme již seznámeni, takže si jen pro pořádek určíme názvy patřičných aktivit a fragmentů. Hlavní aktivitou bude *FeedActivity*, sekundární *FeedDetailActivity* a fragmenty budeme nazývat obdobným způsobem: FeedFragment a *FeedDetailFragment*. Hlavní aktivita bude uživateli poskytovat základní menu,

proto bude potomkem třídy *BaseActivity*. Kvůli aplikovanému návrhovému vzoru si vytvoříme dva odlišné layouty, na něž se budeme odkazovat ukazatelem umístěným ve složce *res/values/layouts*. Nebudeme ale zabíhat do úplných podrobností, postačí nám vědět, že pokud vytvoříme podobné adresáře s trochu rozdílnými jmény např. *res/values-sw600dp-port/layouts* tak na všech zařízeních s uhlopříčkou menší než sedm palců bude zobrazen layout s jedním fragmentem. Na větších zařízeních se bude lépe vyjímat layout se dvěma fragmenty, čímž krásně využijeme jejich velkou plochu. Specifickou skupinou jsou zařízení se sedmipalcovou obrazovkou, při orientaci na výšku nabízejí relativně stále málo místa, ale naležato již dost. Naštěstí pokud bychom se podívali na řešení tvůrců Gmailu, uvidíme v něm eleganci, s jakou takový oříšek rozlouskli. Na výšku zobrazují uživateli jediný fragment a naležato dva. Je to hravé, efektivní a především to zaujme samotné uživatele. My tedy tento účinný princip taktéž přeneseme do naší aplikace.

Co jsme ale zapomněli zmínit je, že *FeedActivity* v obou dvou podobách bude obalovat v ní umístěné fragmenty tzv. *SwipeRefreshLayoutem*. Díky tomu, bude moci uživatel aktualizovat celý obsah jediným pohybem. Stačí, když přejede prstem dolů přes obrazovku, tím se zavolá metoda *loadData()* ve fragmentu *FeedFragment* a celá databáze se nám aktualizuje. Pojďme se tedy na danou metodu podrobněji podívat, abychom zjistili, co se v ní odehrává.

```
if (reloadData) {
            if (NetworkUtils.isNetworkAvailable(getActivity())) {
                new GetRSSFeed().execute(url);
            } else {
                ((FeedActivity) getActivity())
                .setSwipeRefreshingEnabled(false);
                if (Source.isEmpty()) {
                    ((FeedActivity) getActivity())
                    .setSwipeRefreshingEnabled(false);
                    showNetworkInfo();
                } else {
                    ((FeedActivity) getActivity())
                    .setSwipeRefreshingEnabled(false);
                    Toast.makeText(getActivity(),
                            getResources()
                            .getString(R.string.no_network),
                            Toast.LENGTH_SHORT).show();
                }
            }
        } else {
            if (Source.isEmpty()) {
                ((FeedActivity) getActivity())
                .setSwipeRefreshingEnabled(false);
                showNetworkInfo();
            } else {
                initLoader();
            }}}
```

To ona je stěžejním bodem celého fragmentu a rozhoduje, zda všechna data budou nahrazena těmi novějšími z internetu nebo ne. V případech jako je změna orientace

obrazovky nebo běžné otevření této sekce v menu budou data načtena přímo z databáze. Stejně jako v situacích, kdy uživatel sice bude chtít data aktualizovat, ale nebude dostupné žádné internetové připojení. Takže k aktualizaci dojde pouze s aktivním datovým připojením na samotný příkaz uživatele v následujících případech: uživatel přejede u horního okraje obrazovky nebo na začátku seznamu prstem dolů, vybere ve vnořeném menu příkaz k aktualizaci či z něj přidá nový RSS kanál.

A jak tedy celý proces sbírání aktuálních dat probíhá? Ze zdrojového kódu je patrné, že se o něj stará třída *GetRSSFeed* a vykonává ho na pozadí mimo hlavní vlákno, protože je potomkem *AsyncTask*. Každý potomek této třídy zdědí tři elementární metody *onPreExecute()*, *doInBackground()* a *onPostExecute()*. První a poslední slouží k aktualizaci uživatelského prostředí, veškeré dění běžící na pozadí zbývá na starosti zmíněné *doInBackground()*.

Na počátku ještě doplníme zadanou internetovou adresu zadanou jako první parametr, přestože již dříve byla ověřena její podoba pomocí regulárního výrazu *WEB_URL* třídy *Patterns*. Druhý parametr udává název samotného zdroje informací a není povinný. Třída ho obdrží v případech, kdy si uživatel přidává RSS kanál z předem poskytnutého seznamu vybraných orgánů veřejné správy. Tento seznam je vytvořen na základě výsledků provedené analýzy ve druhé kapitole.

Poté smažeme všechny dosud uložené zprávy z databáze příkazem *deleteAll()* volaném na třídě *Item*, protože naše aplikace nebude archivovat starší zprávy, než jaké nabízí samotný zdroj informací. Takovýto způsob řešení v drtivé většině případů naprosto dostačuje. Pro úplnost dodáme, že třída *Item* a *Source* představují odlišné tabulky, takže uložené zdroje informací nám zůstanou v pořádku nedotknuté v druhé tabulce. Nyní tedy vezmeme všechny zdroje a z každého z nich postupně získáme seznam zpráv, jenž obsahuje. To vše za pomoci knihovny *Rome* používající vlastní pomocné třídy a metody optimalizované pro Android platformu. (Android Rome Feed Reader, 2014) Zde je potřeba se na chvíli zastavit a říct, proč jsme zvolili zrovna ji, přestože jsme měli mnohem širší možnosti volby.

Mezi uvažované kandidáty totiž také patřili způsoby spolupracující s třídami *XmlPullParser* či *SaxParser*. Avšak při jejich testování byl u obou zaznamenám celkově horší výkon v porovnání s Rome. V určitých situacích nedokázali získat text zprávy a rychlost zpracování zpráv byla citelně pomalejší. Naproti tomu Rome je dá se říci téměř standard v této oblasti, který si dokáže poradit se všemi záludnostmi spočívající v rozdílu mezi odlišnými specifikacemi RSS 2. 0 a Atom 1.0. Závěrem tohoto rozhodování nakonec byla vcelku logická volba.

Rome tedy svým vlastním způsobem získá veškerý obsah feedu a poskytne nám ho v podobě třídy *SyndFeed*. Z ní si vyseparujeme jen ty informace, o které máme zájem. Třeba název nově přidaného informačního kanálu, ale hlavně v něm obsažené zprávy a jejich evidované podrobnosti. Při tomto procesu se markantním způsobem projevují přednosti knihovny ActiveAndroid, neboť provedení v podobě transakce daný úkon opravdu razantně urychlí. Samozřejmě to záleží na několika okolnostech, ale v porovnání s postupným vkládáním záznamů do databáze je rozdíl doopravdy markantní.

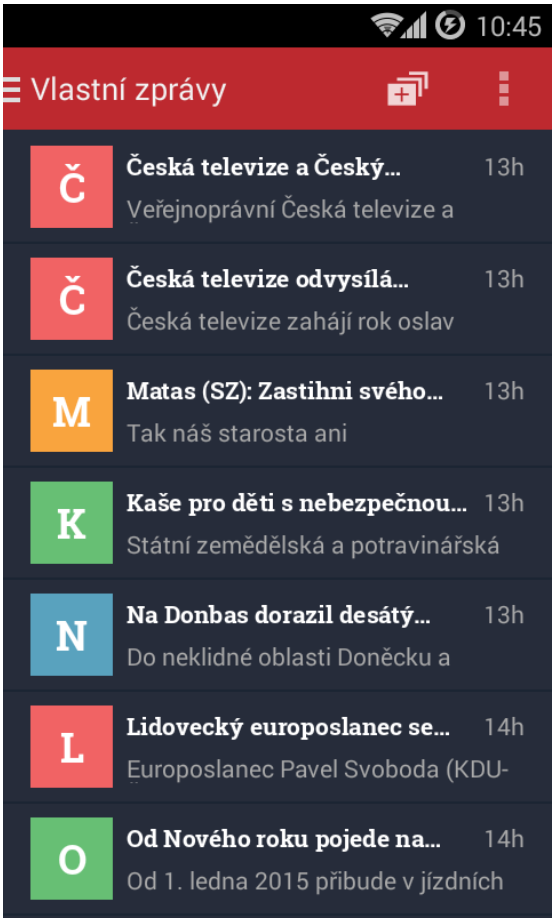
```
private class GetRSSFeed extends AsyncTask<String, Void, Void> {

        @Override
        protected Void doInBackground(String... params) {
            long id = 0;
            if (params[0] != null) {
                Source newSource = new Source();
                if (!params[0].contains("http://")) {
                    newSource.setUrl("http://" + params[0]);
                } else {
                    newSource.setUrl(params[0]);
                }
                if (params[1] != null) {
                    newSource.setName(params[1]);
                } else {
                    newSource.setName(getResources()
                    .getString(R.string.app_name));
                }
                id = newSource.save();
            }
            Item.deleteAll();
            for (Source item : Source.getAll()) {
                    try {
                        URL feedUrl = new URL(item.getUrl());
                        SyndFeedInput input = new SyndFeedInput();
                        SyndFeed feed = input
                        .build(new XmlReader(feedUrl));
                        if (item.getId() == id
                        && params[1].length() == 0) {
                            item.setName(feed.getTitle().trim());
                            item.save();
                        }
                        masterData = feed.getEntries();
                    } catch (Exception e) {
                        if (params[0] != null) {
                            Source.load(Source.class, id).delete();
                        }
                        Log.v("Error Parsing Data", e + "");
                    }
                    try {
                        ActiveAndroid.beginTransaction();
                        for (int j = 0; j < masterData.size(); j++) {
                            Item record = new Item();
                            record.setTitle(masterData.get(j)
                            .getTitle());
                            record.setDescription(masterData.get(j)
                            .getDescription().getValue().trim());
                            record.setLink(masterData.get(j)
                            .getLink());
                            record.setDate(masterData.get(j)
                            .getPublishedDate());
                            record.setSource(item);
                            record.save();
                        }
                        ActiveAndroid.setTransactionSuccessful();
```

```
            } finally {
                ActiveAndroid.endTransaction();
            }
        }
    return null;}
```

Tím pádem jsme získali potřebná data do naší databáze, kde jsou prozatím našim uživatelům k ničemu. K tomu abychom data mohli zobrazit na displeji, nám poslouží třída *CursorLoader*, kterou nám poskytuje samotný Android. Dokáže načíst data z ContentProvideru a ještě za asistence třídy *CursorAdapter* data zobrazit v přehledném seznamu. Inicializace této operace probíhá ve vytvořené metodě *initLoader()*, která je zpravidla volána na konci procesu *GetRSSFeed*. Tedy uvnitř metod *onPostExecute()* a také *loadData()* za předpokladu, že databáze není prázdná. Ukázka zdrojového kódu v tomto případě dost pravděpodobně není nutná.

Nechceme-li uživateli nabídnout obyčejný seznam obsahující položky jedné barvy, který nám nabízí samotná platforma při použití *SimpleCursorAdapteru*, ale něco trochu alespoň zajímavějšího, budeme si muset vytvořit vlastní adaptér. Jeho rodičem bude třída *CursorAdapter* a nazývat ho budeme podobně jako dosud zmíněné fragmenty a aktivity, tedy *FeedAdapter*. Popisovat detaily implementace takové třídy je nošením dříví do lesa, neboť každý mobilní vývojář ví, že tento druh adaptéru nativně používá návrhový vzor ViewHolder a napsat těch pár řádků zdrojového kódu nezabere ani pár minut. My si ale adaptér z výše uvedeného důvodu trochu upravíme podle našich představ. K úpravě použijeme opravdu hezkou knihovnu *TextDrawable*, která stojí za povšimnutí. (TextDrawable, 2014)

Nabízí vlastní prvek uživatelského rozhraní, hodně podobný s tím, jaký používá aplikace Gmail. Prvek má podobu čtverce, jehož barva pozadí je určena hash kódem písmene umístěného uvnitř a lze ho modifikovat vícero způsoby. Přesto zrovna ne takovým, jaký bychom si představovali. Proto si danou knihovnu nebudeme importovat do našeho projektu klasickou cestou za pomoci pluginu Gradle, ale na základě jejího otevřeného zdrojového kódu si vytvoříme totožné třídy a metody. A to vše z jednoho prostého důvodu: v následující sekci aplikace, kde si uživatel bude moci prohlížet program jednání Poslanecké sněmovny, budou uvnitř prvku čísla a kvůli zajištění lepší intuitivnosti celé aplikace, nebudeme barvu pozadí určovat hash kódem, ale podle velikosti čísla. Ale podrobněji si tuto modifikaci knihovny rozebereme až o něco později, kdy se budeme věnovat výše zmíněné sekci. V tuto chvíli nám postačí, když si ukážeme jakým způsobem ji použít doporučeným způsobem.



Obr. 7 Knihovna TextDrawable aplikovaná na seznam zpráv

Při návrhu layoutu, který bude součástí každé položky v seznamu, postačí, když do něj navíc přidáme prvek *ImageView*. Právě v něm bude vykreslen barevný čtverec, o kterém jsme malou chvílí hovořili. Ve zdrojovém kódu si během inicializace třídy *FeedAdapter* nastavíme požadované vlastnosti, jako jsou barva, font a styl písma a následně si v metodě *bindView()* necháme vygenerovat barvu pozadí na základě hash kódu.

```
class FeedAdapter extends CursorAdapter

        private String title, date;
        private ColorGenerator generator = ColorGenerator.LESS;
        private TextDrawable.IBuilder builder = TextDrawable.builder()
                .beginConfig()
                .textColor(Color.WHITE)
                .useFont(Typeface.createFromAsset(getResources()
                .getAssets(), "fonts/robotoslab_bold.ttf"))
                .bold()
                .toUpperCase()
                .endConfig()
                .rect();
```

Další velice užitečnou maličkost, kterou uživateli dopřejeme, bude zobrazení informace, před jakou dobou byla zpráva publikována. Aby výpočet doby nebyl příliš výkonově náročný a nezpomaloval reakce zařízení, zatímco uživatel bude listovat seznamem, vyzkoušíme co nejvíce dostupných variant a porovnáme jen ty nejlepší. Velkým favoritem je knihovna *JodaTime*, která je dokonce lokalizovaná i do češtiny. (Joda-Time, 2002) Přestože její rychlost výpočtu je relativně v pořádku, nevybereme ji, protože nám nevyhovují používaná celá slova jako např. „před hodinou“. Stejný problém se vyskytuje i následující knihovny *PrettyTime*. (PrettyTime, 2014) Přeci jen nechceme, aby tak velkou část zprávy v seznamu zabírala pouze časová informace. Proto se podíváme do zdrojového kódu aplikace Google I/O 2014, protože jsme si vzpomněli, že také ona poskytuje podobnou techniku.

Po důkladném zkoumání jsme nalezli ve třídě *TimeUtil*s metodu s názvem *getTimeAgo()*, jenž řeší výpočet o něco snadněji než předchozí knihovny. Výkon zařízení tedy nebude tolik ovlivněn a časovou informaci lze přizpůsobit našim představám. Díky dostupnosti zdrojového kódu pod licencí Apache 2.0, si vytvoříme v našem projektu totožnou třídu *TimeUtils* a časové údaje zkrátíme na snadno čitelné údaj tvořený číslem a jedním písmenem.

```
public class TimeUtils {

    private static final int SECOND = 1000;
    private static final int MINUTE = 60 * SECOND;
    private static final int HOUR = 60 * MINUTE;
    private static final int DAY = 24 * HOUR;

    public static String getTimeAgo(long time) {
        if (time < 1000000000000L) {
            // if timestamp given in seconds, convert to millis
            time *= 1000;
        }
        long now = Calendar.getInstance().getTimeInMillis();
        if (time > now || time <= 0) {
            return null;
        }
        final long diff = now - time;
        if (diff < 60 * MINUTE)
            return diff / MINUTE + "m";
        else if (diff < 24 * HOUR)
            return diff / HOUR + "h";
        else if (diff < 30L * DAY)
            return diff / DAY + "d";
        else if (diff < 356L * DAY)
            return diff / (30L * DAY) + "M";
        else return diff / (356L * DAY) + "R";
    }}
```

Tím máme *FeedAdapter* téměř kompletní, zbývá nám jen ošetřit, stavy vyvolané v tzv. ActionModu. Máme-li to říct jednoduše co to znamená, tak uživatel si bude moci vybrat vždy jednu položku, nad níž bude možné provádět vybrané operace. V našem případě bude onou operací otevření internetového odkazu uvnitř kon-

krétní zprávy, která půjde provádět daným způsobem pouze v některých případech. Na zařízeních s obrazovkou větší jak sedm palců a sedm palců orientovaných vodorovně se tato operace provede díky vnořenému menu v ActionBaru. Implementací není potřeba podrobněji se zaobírat, snad jen stačí podotknout, že získání odkazu ze zprávy probíhá posunem ukazatele na databázi tzv. Cursoru na aktuální záznam a přečtením hodnoty z vybraného sloupce. Z dřívějších ukázek zdrojového kódu, kde jsme si definovali model databáze, je zřejmé, že odkaz bude uložen ve sloupci *link*. A další věcí k podotknutí je, že metody použité k animaci vybrané položky jsme přizpůsobili tak, aby se bezchybně provedli i na starších verzích API, proto používají animace definované ve složce *res/anim*. Více zabíhat do detailů je pro čtenáře bezcenné, již během vývoje naší aplikace počet uživatelů se zařízeními poháněnými Androidem ve verzích Froyo a Gingerbread rapidně ubývá.

Druhým fragmentem v aktuálně zpracovávané sekci aplikace je *FeedDetailFragment*, jehož účelem je zobrazení kompletní zprávy. Dosud měl možnost uživatel vidět z celé zprávy pouze první řádek nadpisu a krátký úryvek textu. Více informací se do předchozího seznamu ani nevešlo. Protože netušíme, jak moc dlouhé zprávy bude *FeedDetailFragment* zahrnovat, kořenem jeho layoutu zvolíme prvek *ScrollView*. Což uživateli zaručí možnost si prohlížet veškerý obsah zprávy i v případech, kdy se nevejde vcelku na jednu obrazovku.

Problémy ale budou způsobovat obrázky, se kterými jsme z důvodu šetření místa v úložišti ani nepočítali. V seznamu jsme jejich absenci zakryli knihovnou TextDrawable, ale co s nimi v nadcházející situaci? Navíc ne všechny RSS kanály je vkládají do svých zpráv. A ty, které ano je dodávají odlišnými způsoby. První skupina je vkládá standardizovanou cestou v podobě tagu *img* či *media*. S tou si knihovna Rome poradí a nabídne nám obrázky v metodě *getImage()*, které si přesto neuložíme z výše uvedeného důvodu. Druhá skupina je vloží přímo do samotného textu zprávy a to může způsobit nemalé potíže. Pokud bychom totiž text zprávy přiřadili prvku *TextView* metodou *setText()* a jako parametr uvedli řetězec typu *String*, dosáhli bychom opravdu nevzhledného výsledku. Všechny značky a tagy jazyka HTML by totiž neformátovali text, což je jejich primární funkcí, ale zobrazili by se společně s ním. Namísto obrázku by v textu vznikl pouze nějaký čtverec nebo nečitelný znak. Proto jsme nuceni pozměnit typ vkládaného parametru na řetězec třídy *Spannable*, do něhož následně vložíme obsah zprávy následující metodou zaručující korektní výsledek.

```
Spanned spanned = Html.fromHtml(content, new URLImageGetter(text, get-
Activity()), null);
        text.setText(spanned);
```

Získání obrázku z internetu nám obstará třída *URLImageGetter*, jež implementuje rozhraní *ImageGetter* a během jejího vývoje jsme se museli vypořádat s několika zapeklitými nepříjemnosti. Totiž jak změnit velikost samotného prvku *TextView* (v ukázce zdrojového kódu pod proměnnou text), protože jinak obrázek zakryje celý text. Volání metody *invalidate()* na této třídě společně se změnou vlastností *height* a *width* nepřinášel kýžený efekt. Jediným možným východiskem tedy bylo

znovu zavolat metodu *setText()* a výsledek se dostavil okamžitě. Druhou nechtěnou záležitostí byla opravdu rozmanitá škála rozlišení přiložených obrázků. V případě naší vlastní implementace této třídy, naše znalosti nestačili na optimalizaci paměťové náročnosti práce s většími obrázky. Proto jsme byli dotlačeni použít knihovnu *UniversalImageLoader*, jejíž funkčnost je ověřena několika lety vývoje. (Android Universal Image Loader, 2014) Samozřejmě předtím, než s ní začneme pracovat je potřeba jí inicializovat podobně jako třeba Calligraphy.

Obr. 8 Detail zprávy i s obsaženým obrázkem

Jednou z posledních úprav celé této sekce je přidání možností ke správě RSS zdrojů do vnořeného menu. Správou máme na mysli jejich přidávání či odebírání. Během přidávání má uživatel na výběr ze dvou možností, jak danou operaci provést. Buď si vybere ze seznamu předem definovaných zdrojů, které jsou výsledkem provedené analýzy ve druhé kapitole, anebo zadá adresu jakéhokoliv jiného podle vlastního uvážení. V prvním případě stačí během vytváření pole, ze kterého si uživatel bude vybírat, odebrat již uložené zdroje v databázi, tedy záznamy v tabulce, jenž reprezentuje třída *Source*. Ve druhém to bude o něco složitější, protože nemáme zajištěno, zda zadaná internetová adresa je v korektním tvaru. Pro kontrolu tedy použijeme třídou *Patterns* poskytovaný regulární výraz *WEB_URL*. Pokud výraz

kontrolou neprojde, oznámíme to uživateli zobrazení tzv. Toastu. Za to v případě odebírání RSS kanálů toho máme na práci mnohem méně. Díky správně nadefinovaným vztahům mezi tabulkami stačí, když smažeme záznam z tabulky zdrojů a všechny záznamy zprávy s uvedeným zdrojem se také následně odstraní. Tohoto chování je dosaženo díky dříve uvedeným anotacím knihovny ActiveAndroid, konkrétně díky této: *onDelete=Column.ForeignKeyAction.CASCADE*. Jedinou situací, kdy toto řešení nefungovalo, bylo odebrání posledního záznamu z tabulky zdrojů. V tom případě, sice byl uživatel informován o skutečnosti, že databáze je prázdná, ale zároveň třída *CursorLoader* stále poskytovala již smazaná data. Přesnou příčinu se nepovedlo přesně určit, takže jsme po zdlouhavém testování nejrůznějších ošetření přišli na jediné funkční. Před voláním metody *loadData()* zkontrolujeme, zda tabulka zdrojů obsahuje nějaké záznamy příkazem *isEmpty()* volaném na třídě *Source* a v kladném případě nahradíme celý fragment za nový pomoci třídy *SupportFragmentManager* a transakce *replace()*.

Eliminací těchto drobných nepříjemností jsme tedy zkompletovali celou sekci tvořenou aktivitami FeedActivity, FeedDetailActivity dohromady s fragmenty *FeedFragment*, *FeedDetailFragment*. V menu si ji označíme názvem „Vlastní zprávy“.

4.3 Použití základní struktury

Nyní můžeme drtivou většinu použitých principů a metod aplikovat i na ostatní podobné sekce, s tím rozdílem, že každá bude mít určená svůj jediný zdroj informací, který uživatel nebude moci žádným způsobem změnit. Konkrétně se jedná o sekce korespondující s výsledky analýzy provedené ve druhé kapitole této práce, jejichž názvy jsou „Prezident ČR – Tiskové zprávy“, „Poslanecká sněmovna – Nové sněmovní tisky“ a „Vláda ČR – Tiskové informace“.

Při rozšiřování aplikace o uvedené sekce však budeme muset zároveň přidat nové tabulky do naší databáze. Důvod, který nás k tomu nutí, je prostý. Jelikož u těchto zpráv nepotřebujeme evidovat jejich zdroj, postačí, když si pro každou novou sekci vytvoříme tabulky s příznačným jménem. To nám umožní ověřovat skutečnost, zda je či není tabulka prázdná stejnou metodou jako doposud, tedy *isEmpty()* a ušetří nám čas strávený psaním dalších SQL dotazů a jejich následné testování nejčastěji prováděné za pomoci nástroje adb. Pro úplnost tedy zmíníme názvy tříd definující jednotlivé tabulky, které jsou téměř identické jako u třídy *Item*: *ItemPresident*, *ItemParliament* a *ItemGovernment*. Jediný rozdíl mezi nimi a třídou *Item* je, že nejsou v žádném vztahu s ostatními tabulkami, ani s třídou *Source*. Tím se nám aplikace pěkně rozrostla, a díky předtím dokončené a odladěné sekci, byl tento krok relativně jednoduchý. Jediný významný rozdíl v implementaci se odehrál uvnitř třídy *GetRSSFeed*, přesněji v metodě *doInBackground()*, kde jsme odstranili *for cyklus* pro načítání zdrojů z databáze. Místo něj získáváme zprávy pouze z jedné napevno určené adresy.

Aby měl uživatel přeci také jen k dispozici ucelený přehled nejnovějších zpráv z nově vytvořených sekcí s pevně určenou adresou, vytvoříme pro něj souhrn novinek. Na pomoc si vezmeme knihovnu *ProgressWheel*, díky níž vytvoříme snadno

ovladatelné rozhraní ve formě tlačítka. (Progress Wheel, 2014) Nebudeme se zde zdržovat zbytečně dlouhým popisem, jak to vlastně celé vypadá a prohlédneme si hotovou podobu sekce nazývané „Aktuality“ na obrázku č. 9. K tomu je nutné snad jen dodat, že ovládání probíhá buď gesty provedené prstem doprava či doleva anebo stiskem tlačítka, jenž zastaví odpočet času, za který bude aktuální zpráva na obrazovce nahrazena za další.

Obr. 9 Sekce „Aktuality“

Je jasné, že z obrázku nemůžeme vyčíst logiku, na které to celé funguje, takže si ji teď trochu detailněji popíšeme a vysvětlíme. Není to nic složitého, dá se tvrdit, že vše řídíme pomocí jedné proměnné typu *boolean*, jejíž budeme nazývat *paused*. Při událostech jako stisknutí tlačítka bílé barvy nebo provedení gesta přes nadpis zprávy změníme její současnou hodnotu na obrácenou. Toho si všimne vedlejší vlákno spuštěné při samotném vzniku daného fragmentu, tedy uvnitř metody *onCreateView()*. Zkontroluje pozměněnou hodnotu proměnné *paused* a adekvátně zareaguje, tzn. změní obrázek vykreslovaný přes bílé tlačítko. Pro tento šikovný trik je nutné zvolit kořenem layoutu prvek *FrameLayout*, jehož vlastnosti nám to umožní. Pořadí v něm umístěných prvků totiž určuje vrstvu jejich zobrazení. Jednoduše řečeno poslední prvek bude překrývat ty předchozí.

Ke změně zprávy dojde, jak jsme si již řekli, buď vypršením časového limitu anebo pohybovým gestem. Časový limit nám odměřuje samotné tlačítko knihovny ProgressWheel, z kterého získáme metodu *getProgress()* číslo v rozmezí od 0 do 360. Pokud nám tedy bude tlačítko vracet nejvyšší možnou hodnotu, necháme animovat změnu zprávy. Tuto operaci opět zajišťuje vedlejší vlákno. Jediný případ, kdy má na starosti danou operaci někdo jiný, je během rozpoznávání pohybových gest. Dosud jsme nezmínili, že animace zpráv staré za nové, je prováděna prvkem *TextSwitcher*. A právě jemu přiřadíme instanci třídy *TouchListener* společně se *SimpleGestureDetectorem*. Díky dostupným metodám *onFling()* a *onSimpleTapUp()* můžeme jednoduchým způsobem detekovat nejrůznější gesta platformy Android. První bude rozpoznávat delší vodorovný tah prstem a druhá obyčejné kliknutí, protože souběžná aplikace *TouchListeneru* a standardní metody *onClick()* není jednoduše možná. V ukázce zdrojového kódu si lze také všimnout během identifikace délky vodorovného tahu, nezbytného přepočtu tzv. density, tedy hustoty obrazových bodů na jeden palec. Maximální i minimální délku jsme si definovali vlastní, neboť systémem definované hodnoty nám jednoduše nepozdávali.

```
final GestureDetector gesture = new GestureDetector(getActivity(),
                new GestureDetector.SimpleOnGestureListener() {

                    // Get the screen's density scale
                    final float scale = getResources()
                    .getDisplayMetrics().density;
                    final int SWIPE_MIN_DISTANCE = (int)(50.0f*scale);
                    final int SWIPE_MAX_OFF_PATH =(int)(250.0f*scale);
                    final int SWIPE_THRESHOLD_VELOCITY =
                        (int)(100.0f*scale);

                    @Override
                    public boolean onDown(MotionEvent e) {
                        return true; }

                    @Override
                    public boolean onSingleTapUp(MotionEvent e) {
                        // Build the intent
                        Uri webpage = Uri.parse(db.get(activeIndex)
                        .get(TAG_DETAIL));
                        Intent webIntent = new Intent(Intent
                        .ACTION_VIEW, webpage);
                        startActivity(webIntent);
                        return true;
                    }

                    @Override
                    public boolean onFling(MotionEvent e1,
                        MotionEvent e2, float velocityX, float veloci-
                        tyY) {
                              try {
                            if (Math.abs(e1.getY() - e2.getY()) >
                            SWIPE_MAX_OFF_PATH)
                                return false;
```

```
                    if (e1.getX() - e2.getX() >
                    SWIPE_MIN_DISTANCE
                    && Math.abs(velocityX) >
                    SWIPE_THRESHOLD_VELOCITY) {
                            changeText(false);
                    } else if (e2.getX() - e1.getX() >
                    SWIPE_MIN_DISTANCE
                    && Math.abs(velocityX) >
                    SWIPE_THRESHOLD_VELOCITY) {
                            changeText(true);
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
                return super.onFling(e1, e2, velocityX,
                velocityY);
            }
        });
wText.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        return gesture.onTouchEvent(event);
    }
});}
```

Načítání zpráv z databáze je zde naprosto primitivní, získané údaje si uchováváme v seznamu o 15 položkách. To znamená, že z nejnověji zkonstruovaných sekcí, tedy těch s pěvně určenou adresou zdrojů, si vybereme nejčerstvějších 5 zpráv. Pozici aktuální položky máme v samostatné proměnné a podle směru pohybu ji buďto zvyšujeme či snižujeme. Samo sebou máme ošetřené případy objevující se na začátku a konci seznamu.

Dosud jsme také neprozradili skutečnost o tom, že velikost písma v prvku *TextSwitcher* je spočítaná na základě délky řetězce a dokáže se přizpůsobit. Takovou funkcionalitu samotný Android nenabízí a během vývoje nebyla dostupná žádná knihovna, která by to dokázala. Což už pár týdnů neplatí, protože se objevila knihovna s názvem *Android-AutoFitTextView*. (AutoFitTextview, 2014) My si tedy tuto vymoženost v naší aplikaci implementovali vlastnoručně. Založili jsme si novou třídu *AutoResizeTextView* dědící vlastnosti od *TextView* a *for cyklem* zkoušeli všechny možnosti od největšího písma po nejmenší. Hranice velikostí fontu jsme si definovali s ohledem na orientaci a rozlišení displeje. Možná bude lepší v budoucích aktualizacích přejít k uvedené knihovně, protože komunita ji dokáže rychleji vylepšovat a hlavně ověřovat její funkčnost na větším počtu zařízení než máme my k dispozici. Ale přejděme již dále.

Na obrázku č. 9 si lze také povšimnout tlačítka nesoucí logo společnosti, kterou má naše aplikace propagovat. V souladu s navrženou marketingovou strategií tedy umístíme do aplikace obrazovku s jejími doprovodnými informacemi. Aby byla samotná prezentace o něco málo zajímavější, přizpůsobíme způsob prohlížení podle velikosti zařízení. Na telefonech spustíme novou aktivitu, za to na tabletech zobrazíme menší dialog. Myšlenka je to bezesporu výborná, ale jak ji zrealizovat?

Aplikovaná knihovna SupportLibrary konkrétně appcompat-v7 umožňuje použít na aktivitu styl *DialogWhenLarge*, jenže tato varianta funguje až od API úrovně 11. Tady se projevuje fakt, že s podporou starších zařízení se už prostě nepočítá. Anebo to technologie ani neumožňuje, což naneštěstí není náš případ. Pomůže nám třída *DialogFragment*, která jde zobrazit jak uvnitř aktivity, tak jako samostatný dialog.

```
public void openInfo(View v) {
        if ((Configuration.SCREENLAYOUT_SIZE_MASK) != Configuration
        .SCREENLAYOUT_SIZE_NORMAL)
        && (Configuration.SCREENLAYOUT_SIZE_MASK != Configuration
        .SCREENLAYOUT_SIZE_SMALL)) {
            showPopup(v);
        } else {
            startActivity(new Intent(getActivity(), InfoActivity
            .class));
        }}
```

4.4 Spolupráce aplikace s API

Tímto je tedy sekce „Aktuality“ kompletně dokončena a my přejdeme k předposlední části celé aplikace, jejíž název zní „Program jednání Poslanecké sněmovny“. U ní budeme znovu aplikovat návrhový vzor Master/detail dohromady s postupy vyvinutými při tvorbě úplně první části „Vlastní zprávy“, ovšem v pozměněné podobě. Nejrazantnější změnou bude absence *CursorAdapteru*, který je zde zbytečný hned z několika důvodů.

Vrátíme-li se o několik kapitol zpátky, kde jsme si definovali strukturu našich dat, tak si připomeneme druhý typ dat ve formě souboru JSON. Také jsme si řekli, že ho budeme ukládat v nepozměněné podobě do stejného úložiště, na kterém je nainstalována sama aplikace. Z toho vyplývá, že tentokrát nebudeme pracovat s databází, takže u seznamu použijeme obyčejný adaptér s rodičem třídy *BaseAdapter*. Implementace je hodně podobná a navíc nemáme tolik prostoru, abychom si ji zde celou ukazovali.

Mnohem podstatnější je způsob, jakým získáme daný soubor. Ten jsme si sice teoreticky předvedli, ale bylo by dobré si ho rozebrat více do hloubky. Pojďme se tedy do toho pustit a zjistit jak je webové API naprogramováno. Abychom se vyhnuli jakýmkoliv pochybnostem hned na začátku, řekneme si, že samotné API je napsáno v programovacím jazyce PHP.

Jeho REST architektura nám umožní všechny běžné CRUD operace pro přístup ke zdrojům, ale v tomto případě bohatě postačí implementace metod pro získání zdrojů, tedy *GET*. Podle toho jak formulujeme požadavek, budou volány konkrétní metody. První, kterou si představíme, bude *saveToFile()*. Jak název napovídá její funkcí, bude uložení dat předaných v parametru do cache souboru. Ještě než se si analyzujeme způsob, kterým předávané data získáme, podotkneme, že *saveToFile()* je jednou denně ze strany serveru volána Cronem. Program jednání, se zase tak často neměnní, proto jsme zvolili jednodenní časový interval aktualizace.

```
<?php
header('Content-type: text/json;charset=UTF-8');
require_once 'parserController.php';

if(isset($_GET["action"])){
      switch($_GET["action"]){
            case "save":
                  ParserController
                  ::saveToFile(ParserController::getAll());
            break;
            case "getall":
                  echo ParserController::getDataFromFile();
            break;
      }}?>
```

Ústředním bodem je bezpochyby metoda *getAll()* nalézající se uvnitř třídy *ParserController*. Ta vrací objekt typu JSON, který můžeme rovnou ukládat. A to bez dodatečných úprav, neboť JSON je datový formát nezávislý na počítačové platformě. Je považovaný za odlehčenou verzi XML, také díky tomu, že sází na jednoduchost způsobu uložení dat, srozumitelnost, jednotnost a to vše na úkor velikosti přenášených dat. Datová struktura samotného objektu je závislá na obsažených datech, v našem případě je její výsledná podoba zobrazená v příloze C.

Po pečlivém zkoumání hierarchie webových stránek Poslanecké sněmovny jsme tedy navrhli pomocnou metodu *getMeetingList()*, díky níž získáme ze zdrojového HTML kódu potřebné informace v definované struktuře. Její zdrojový kód je natolik rozsáhlý a složitý, že případné zájemce o jeho přečtení rovnou odkážeme do přílohy D, konkrétně do souboru s názvem *class_parser.php*. Ale vraťme se zpět k předchozí ukázce. Výsledný zdroj získáme přidáním parametru *getAll()* nakonec URI, kterou bude aplikace načítat za pomocí tzv. streamů a třídy *HttpURLConnection* uvnitř fragmentu *ProgramFragment*. Ten je hodně podobný s *FeedFragmentem* v části aplikace nazývané „Vlastní zprávy“.

Kvůli úspoře dat přenášených v internetové síti budeme požadavek odesílat pouze při prvním spuštění aplikace a následně pouze na povel uživatele. Jakmile totiž budeme mít soubor uložený v daném zařízení, můžeme jej rovnou načítat odsud. Což je užitečné i v situacích, kdy uživatel zrovna není připojen k internetu.

Další užitečnou věc, kterou aplikace bude nabízet, je připomenutí některého bodu z programu Poslanecké sněmovny. Výběr bodu neboli jednání bude proveden tím samým způsobem, jako v předešlých sekcích akce otevření zprávy ve webovém prohlížeči tzn. kliknutím na barevný čtverec knihovny TextDrawable uvnitř každé položky anebo pokud čte detail jednání, tak za pomoci vnořeného menu. Přesný čas připomenutí si uživatel zvolí na jakékoliv verzi Androidu stejným způsobem díky knihovně *ReminderDatePicker*, protože dokáže importovat číselník z novějších verzí systému i na ty starší. (ReminderDatePicker, 2014)

Poslední odlišností od předchozích sekcí bude zobrazení detailu jednání, který je narozdíl od relativně krátké RSS zprávy, někdy hodně dlouhý. To protože obsahuje tzv. body k projednání a jejich počet někdy přesahuje i jednoho sta. Kvůli nim nebude tedy hlavním prvkem layoutu *ScrollView*, ale seznam. Místo obvyklého

ListView, ale vybereme knihovnu *StickyListHeaders*. Na horním okraji obrazovky proto bude vždy zobrazen aktuální nadpis, aby měl uživatel vždycky přehled, kde se aktuálně nachází. Specifické metody implementované uvnitř samotné třídy *ProgramAdapter* si lze prohlédnout v následující ukázce. Vzhledem k některým extrémním délkám seznamu s těmito položkami při návrhu layoutu také přidáme atribut *fastScroll*, díky němuž uživatel bude moci listovat rychleji.

```
@Override
public View getHeaderView(int position, View convertView, ViewGroup
parent) {
      HeaderViewHolder headerHolder = null;
      if (convertView == null) {
            headerHolder = new HeaderViewHolder();
            convertView = mInflater
            .inflate(R.layout.fragment_detail_header,
            parent, false);
            headerHolder.textView = (TextView) convertView
            .findViewById(R.id.header);
            convertView.setTag(headerHolder);
      } else {
            headerHolder = (HeaderViewHolder) convertView.getTag();
      }
      headerHolder.textView
      .setText(data.get(position).get(TAG_TITLE));
      return convertView;
}

@Override
public long getHeaderId(int position) {
      return Long.parseLong(data.get(position).get(TAG_TITLE_ID));}
```

4.5 Přehrávání videa

A jednou z nejobtížnějších sekcí, kterou nám zbývá dokončit, je ta s videem jednání Poslanecké sněmovny. Původně jsme sice chtěli rozšířit možnosti našeho webového API o detekci, zda je či není video na příslušných webových stránkách dostupné, ale po dlouhodobějším pozorování jsme došli k závěru, že se to prostě nevyplatí. Video zde bývá možné najít i v době, kdy vysílání neprobíhá. Takže kontrolu dostupnosti převezme samotná aktivita s názvem *VideoPlayerActivity*, která bude spuštěna jen ve chvílích, kdy bude mít zařízení přístup k internetu. V opačném případě bude uživatel požádán, aby se připojil do sítě.

Během sestavování požadavků, která by měla aplikace zvládat, jsme si vybrali dvě knihovny, které jsme považovali za schopné se vypořádat s přehráváním daného videa. První a vhodnější volbou se nám zdál projekt Vitamio, který se snadněji integruje. Bohužel ani on během testování zda, video přehraje či ne, neobstál. Proto jsme přešli k poslední variantě, kterou nám poskytl open-source přehrávač VLC. Ten tedy jako jediný dokázal video spolehlivě přehrát. Ovšem kvůli podpoře snad všech druhů operačních systémů je celý napsán v programovacím jazyce C++, což se při nasazení do naší aplikace neobejde bez potíží. Android používá totiž

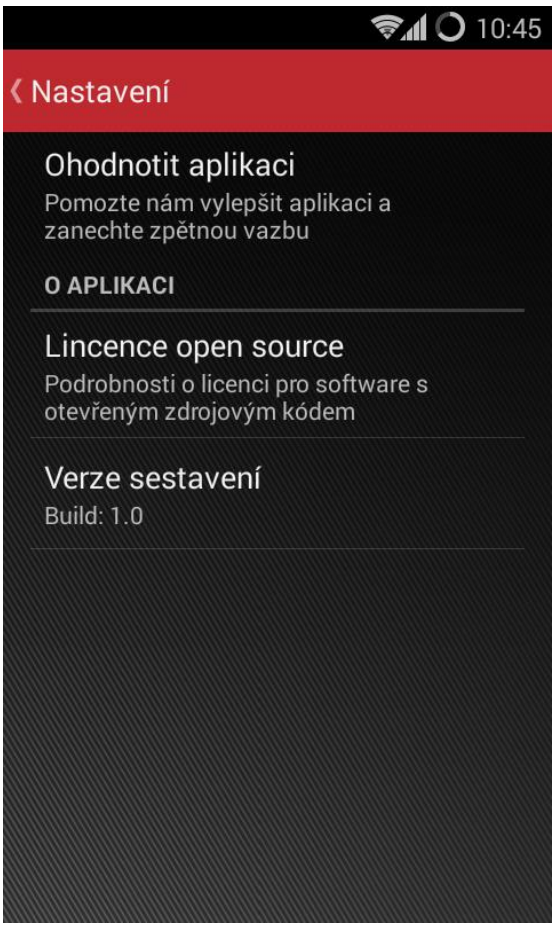
programovací jazyk Java a programy napsané v jazyce C++ musí být do aplikace integrovány za pomoci doplňkového vývojového prostředí NDK. To lze snadno propojit s jakýmkoliv Java editorem, tedy i Android Studiem.

Problém, který se nám nepovedlo úspěšně celý rozlousknout, však nastal během kompilace zdrojových souborů knihovny VLC prováděné podle *návodu*. (android Compile, 2011) Kvůli rozdílné architektuře procesorů pohánějících zařízení s Androidem, je nutné provést kompilaci pro každý druh zvlášť. Což se povedlo pouze architektury typu ARMv7. U ostatních totiž proces neproběhl až do konce z neznámé příčiny. To v konečném důsledku znamená, že přehrávání videa bude funkční pouze na již zmíněném typu procesorů. Na zařízeních s odlišným typem pouze uživatele informujeme, že jeho zařízení dosud tuto funkcionalitu nepodporuje a vyřeší to pozdější aktualizace.

Při implementaci třídy *VideoPlayerActivity* se budeme držet dostupného *příkladu*. (LibVLC Android Sample, 2014) A přizpůsobíme si jej v několika ohledech. Především zde chybí naslouchání situacím, kdy video není dostupné.

```
private Handler mHandler = new MyHandler(this);

    private static class MyHandler extends Handler {
        private WeakReference<VideoPlayerActivity> mOwner;
        public MyHandler(VideoPlayerActivity owner) {
            mOwner = new WeakReference<VideoPlayerActivity>(owner);
        }

        @Override
        public void handleMessage(Message msg) {
            VideoPlayerActivity player = mOwner.get();
            if (msg.what == VideoSizeChanged) {
                player.setSize(msg.arg1, msg.arg2);
                return;
            }

            // Libvlc events
            Bundle b = msg.getData();
            switch (b.getInt("event")) {
                case EventHandler.MediaPlayerEncounteredError:
                    player.releasePlayer();
                    player.showError();
                    break;
                case EventHandler.MediaPlayerEndReached:
                    player.releasePlayer();
                    player.finish();
                    break;
                case EventHandler.MediaPlayerPlaying:
                    player.mProgress.setVisibility(View.INVISIBLE);
                    break;
                case EventHandler.MediaPlayerPaused:
                case EventHandler.MediaPlayerStopped:
                default:
                    break;
            }}}
```

Tímto se tedy dá celá aplikace považovat za dokončenou, předtím než si však vygenerujeme instalační balíček určený k distribuci, doladíme poslední maličkosti. Přidáme tedy aktivitu s názvem Nastavení. Do ní umístíme možnost ohodnotit aplikaci, která přesměruje uživatele přímo do obchodu Google Play a také tlačítko k zobrazení licenčních dodatků, ke každé integrované knihovně. V souboru AndroidManifest.xml určíme jako výchozí spouštěnou aktivitu *SplashActivity*, která při každém startu aplikace po dobu tří vteřin zobrazí logo. Hotový instalační balíček je také nutno podepsat certifikátem tzv. keystore, který si může každý vývojář vygenerovat pomoci standardních nástrojů prostředí JDK.



Obr. 10 Sekce „Nastavení“

5 Distribuce

Po úspěšném zvládnutí všech nepříjemností spojených s vývojem a testováním přichází na řadu výběr vhodné varianty distribuce mezi uživatele. Možností, jak v současné době poskytovat Android aplikaci je celá řada, takže si představíme alespoň ty nejběžnější.

5.1 Google Play

Společně s konkurenční službou App Store od společnosti Apple je jednou z největších a nejdůležitějších distribučních kanálů mobilních aplikací. Přestože je primárně zaměřen na aplikace pro chytré mobilní telefony a tablety s Androidem, tak nabízí i několik dalších druhů digitálního obsahu jako např. hudbu, knihy a filmy. Nabízí mnoho užitečných nástrojů napomáhajících se samotnou distribucí aplikace pro cílovou skupinu uživatelů a sledováním nejrůznějších statistik, které můžeme využít k dalšímu vylepšování aplikace.

Díky tomu, že k tomuto způsobu distribuce aplikací má přístup snad každý uživatel zařízení s Androidem, jej využijeme u naší aplikace. Pro zveřejnění v Google Play je potřeba mít přístup do tzv. Developer Console, jenž je zpoplatněn jednorázovým poplatkem ve výši 25$. Naštěstí pro tento účel použijeme Google účet společnosti, ve které jsem vykonával pracovní stáž a má do Developer Console přístup již zpřístupněn. Před zveřejněním omezíme dostupnost aplikace pro Českou a Slovenskou republiku, překlady do ostatních jazyků jsme neprováděli a vzhledem k celkovému zaměření na veřejnou správu České republiky by se to pravděpodobně ani nevyplatilo.

5.2 Alternativní možnosti

Další významná služba je Amazon Appstore, jež zatím bohužel nemá pro nás tak perspektivní počet uživatelů. Mluvíme samozřejmě o regionu Střední Evropy, neboť v celosvětovém měřítku je její působnost značná a dále roste opravdu rychlým tempem. Jedním z důvodu proč není v České či Slovenské republice tolik oblíbená, je pravděpodobně chybějící lokalizace a také dostupnost příslušné mobilní aplikace pouze na jejich webových stránkách.

Tím se dostáváme k poslední uvažované možnosti prezentace v podobě vlastních webových stránek. Ta však v současné době není příliš vhodným řešením, pokud nemá dostatečný počet návštěvníků. Amazon Appstore je toho jasným důkazem. Navíc Google Play díky jeho licenčním podmínkám zabrání neoprávněným instalacím a použití aplikace.

6 Závěr

Android aplikaci se podařilo zdárně dokončit a umístit na Google Play, kde je volně k dispozici na následující webové adrese https://play.google.com/store/apps/details?id=eu.imakers.asistent. Tím tedy byl úspěšně splněn hlavní cíl práce, i přes drobné nedostatky týkající se přehrávání videa. Vzhledem k problémům spojených s kompilací zdrojových souborů knihovny VLC, je aplikace schopna zajistit přehrávání videa pouze na zařízeních s procesory postavenými na architektuře ARMv7. Kvůli vybraným informacím, konkrétně programu jednání Poslanecké sněmovny, které nejsou poskytovány žádným RSS kanálem, bylo vyvinuto vlastní webové REST API. Samozřejmě není univerzální, ale lze snadno rozšířit a použít i u dalších podobných projektů. Nepříjemnosti doprovázející různorodost specifikací RSS technologie se v aplikaci nijak neprojevují a obsah zprávy je vždy zobrazen kompletní nezávisle na tom, zda je označen specifikací RSS 2.0 či Atom 1.0. Rovněž se povedlo zajistit zpětnou kompatibilitu aplikace dokonce i s Androidem ve verzi 2.2 Froyo, což v porovnání s ostatními aplikacemi představuje stále obrovskou výhodu. Rozvržení ovládacích prvků je vždy přizpůsobeno velikosti daného zařízení, jednoduše řečeno uživatel si přečte zprávy stejně pohodlně jak na tabletu, tak na chytrém telefonu.

Implicitně nabízené RSS zdroje, které si může uživatel dodatečně přidat a sledovat, korespondují s výsledky provedené analýzy zaměřené na preference uživatelů. Na základě těchto výsledků také byla stanovena marketingová strategie, která sice byla realizována, ale vzhledem ke zdržením během vývoje a nedávnému datu vydání aplikace ji není možné objektivně zhodnotit.

6.1 Možnosti dalšího rozšíření

Na první místo seznamu věcí k vylepšení rozhodně patří rozšíření podpory přehrávání videa na ostatních typech procesorů typu: x86, MIPS, ARMv6, ARM v5. Což je jednoznačně největší priorita, další věcí na seznamu je přidání podpory nové verze Androidu 5.0, jenž byla vydána během vývoje aplikace a přináší opravdu zásadní změny jak v designu aplikací, tak i v jejich výkonu. Na tom jde názorně vidět, jak rychle jde vývoj celé Android platformy kupředu.

To jsou tedy asi ty nejpalčivější problémy a další vylepšování aplikace by mohlo probíhat na základě zpětné vazby a připomínek uživatelů.

7 Literatura

ActionBarSherlock. *GitHub* [online]. 2014 [cit. 2014-12-28]. Dostupné z: https://github.com/JakeWharton/ActionBarSherlock

ActiveAndroid. *GitHub* [online]. 2014 [cit. 2014-12-28]. Dostupné z: https://github.com/pardom/ActiveAndroid

ALLEN, Grant. *Android 4: průvodce programováním mobilních aplikací.* 1. vyd. Překlad Jakub Mužík. Brno: Computer Press, 2013, 656 s. ISBN 978-80-251-3782-6.

Android-Bootstrap. *GitHub* [online]. 2014 [cit. 2014-12-28]. Dostupné z: https://github.com/Bearded-Hen/Android-Bootstrap

AndroidCompile. *VideoLAN Wiki* [online]. 2011, 2014 [cit. 2014-12-28]. Dostupné z: https://wiki.videolan.org/AndroidCompile

Android Developers Guide [online]. 2011, 2014 [cit. 2014-05-11]. Dostupné z: http://developer.android.com/develop/index.html

Android I/O Android App. *GitHub* [online]. 2014 [cit. 2014-12-28]. Dostupné z: https://github.com/google/iosched

Android ROME Feed Reader. *Google Project Hosting* [online]. 2014 [cit. 2014-12-28]. Dostupné z: https://code.google.com/p/android-rome-feed-reader/

Android, the world's most popular mobile platform. *Android Developers* [online]. 2011, 2014 [cit. 2014-05-11]. Dostupné z: http://developer.android.com/about/index.html

Android Universal Image Loader. *GitHub* [online]. 2014 [cit. 2014-12-28]. Dostupné z: https://github.com/nostra13/Android-Universal-Image-Loader

App Structure. *Android Developers* [online]. 2011, 2014 [cit. 2014-12-16]. Dostupné z: http://developer.android.com/design/patterns/app-structure.html

AutoFitTextView. *GitHub* [online]. 2014 [cit. 2014-12-28]. Dostupné z: https://github.com/grantland/android-autofittextview

Bulding a Flexible UI. *Android Developers* [online]. 2011, 2014 [cit. 2014-12-27]. Dostupné z: http://developer.android.com/training/basics/fragments/fragment-ui.html

BURTON, MICHAEL A FELKER DONN. *Android application development for dummies*. 2nd edition. xviii, 386 pages. ISBN 978-1-118-38710-8.

BusinessInfo.cz [online]. 1997, 2014 [cit. 2014-12-28]. Dostupné z: http://www.businessinfo.cz

Calligraphy. *GitHub* [online]. 2014 [cit. 2014-12-28]. Dostupné z: https://github.com/chrisjenx/Calligraphy

Dashboards: Platform Versions. *Android Developers* [online]. 2011, 2014 [cit. 2014-11-25]. Dostupné z: http://developer.android.com/about/dashboards/index.html

ExoPlayer. *GitHub* [online]. 2014 [cit. 2014-12-28]. Dostupné z: https://github.com/google/ExoPlayer

Feedly. Your news reader. *Google Play* [online]. 2012 [cit. 2014-12-28]. Dostupné z: https://play.google.com/store/apps/details?id=com.devhd.feedly

FFmpeg. *GitHub* [online]. 2014 [cit. 2014-12-28]. Dostupné z: https://github.com/FFmpeg/FFmpeg

Genymotion [online]. 2014 [cit. 2014-12-27]. Dostupné z: https://www.genymotion.com/#!/

GreenDAO. *GitHub* [online]. 2014 [cit. 2014-12-28]. Dostupné z: https://github.com/greenrobot/greenDAO

HEROUT, PAVEL. *Učebnice jazyka Java.* 5. rozš. vyd. České Budějovice: Kopp, 2010, 386 s. ISBN 978-80-7232-398-2.

LibVLC Android Sample. *Bitbucket* [online]. 2014 [cit. 2014-12-28]. Dostupné z: https://bitbucket.org/edwardcw/libvlc-android-sample/overview

Joda-Time [online]. 2002, 2014 [cit. 2014-12-28]. Dostupné z: http://www.joda.org/joda-time/

MURPHY, MARK L. *Android 2: průvodce programováním mobilních aplikací.* Vyd. 1. Brno: Computer Press, 2011, 375 s. Průvodce (Grada). ISBN 978-80-251-3194-7.

Navigation Drawer. *Android Developers* [online]. [cit. 2014-12-27]. Dostupné z: http://developer.android.com/design/patterns/navigation-drawer.html

Parlament České republiky [online]. 1995, 2014 [cit. 2014-12-28]. Dostupné z: http://www.psp.cz/

ParlamentniListy.cz [online]. 2009, 2014 [cit. 2014-12-28]. Dostupné z: http://www.parlamentnilisty.cz/

Pražský hrad [online]. 2014 [cit. 2014-12-28]. Dostupné z: http://www.hrad.cz/

PrettyTime [online]. 2014 [cit. 2014-12-28]. Dostupné z: http://ocpsoft.org/prettytime/

Progress Wheel. *GitHub* [online]. 2014 [cit. 2014-12-28]. Dostupné z: https://github.com/Todd-Davies/ProgressWheel

ReminderDatePicker. *GitHub* [online]. 2014 [cit. 2014-12-28]. Dostupné z: https://github.com/SimplicityApks/ReminderDatePicker

StickyListHeaders. *GitHub* [online]. 2014 [cit. 2014-12-28]. Dostupné z: https://github.com/emilsjolander/StickyListHeaders

Sugar ORM. *GitHub* [online]. 2014 [cit. 2014-12-28]. Dostupné z: https://github.com/satyan/sugar

Support Library. *Android Developers* [online]. 2011, 2014 [cit. 2014-12-28]. Dostupné z: http://developer.android.com/tools/support-library/index.html

TextDrawable. *GitHub* [online]. 2014 [cit. 2014-12-28]. Dostupné z: https://github.com/amulyakhare/TextDrawable

UJBÁNYAI, MIROSLAV. *Programujeme pro Android.* Vyd. 1. Praha: Grada, 2012, 187 s. Průvodce (Grada). ISBN 978-80-247-3995-3.

Vitamio for Android. *GitHub* [online]. 2014 [cit. 2014-12-28]. Dostupné z: https://github.com/yixia/VitamioBundle/

Vláda ČR [online]. 2009, 2014 [cit. 2014-12-28]. Dostupné z: http://vlada.cz/

VLC media player. *GitHub* [online]. 2014 [cit. 2014-12-28]. Dostupné z: https://github.com/videolan/vlc

VOLEK, PETR A PŘENOSIL JAN. *Veřejná správa.* Vyd. 1. V Brně: Mendelova zemědělská a lesnická univerzita, 2005. ISBN 80-715-7847-9.

Přílohy

A Dotazník

Vážený respondente, vážená respondentko

V rámci své bakalářské práce si Vás dovoluji požádat o vyplnění dotazníku, který je určený každému, kdo vlastní alespoň jeden chytrý mobilní telefon či tablet s operačním systémem Android. Cílem práce je Android aplikace pro sledování novinek a aktualit z veřejné správy, jež obsahuje velké množství nejrůznějších institucí. Dotazník si tedy klade za cíl zjistit, o které je největší zájem.
Veškeré odpovědi jsou anonymní.

Jan Jacko, student PEF MENDELU

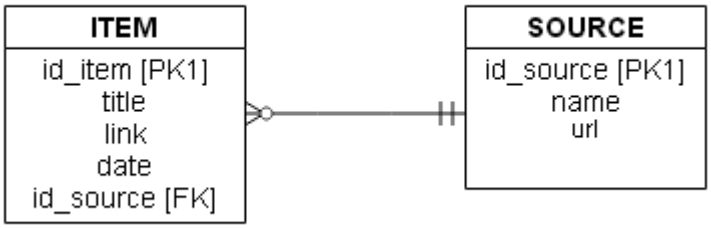
1. Které 3 oblasti by Vás v případě zájmu o dění ve veřejné správě zajímaly nejvíce?
   - Prezident ČR (http://www.hrad.cz)
   - Vláda ČR (http://www.vlada.cz)
   - Senát ČR (http://www.senat.cz)
   - Poslanecká sněmovna (http://www.psp.cz)
   - žádné

2. Kde dále byste čerpal/a informace o veřejné správě?
   - Parlamentní listy (http://www.parlamentnilisty.cz)
   - Business Info (http://www.businessinfo.cz)
   - nezajímá mě to
   - jiné (uveďte které konkrétně)

     _____

B ERD diagram databáze

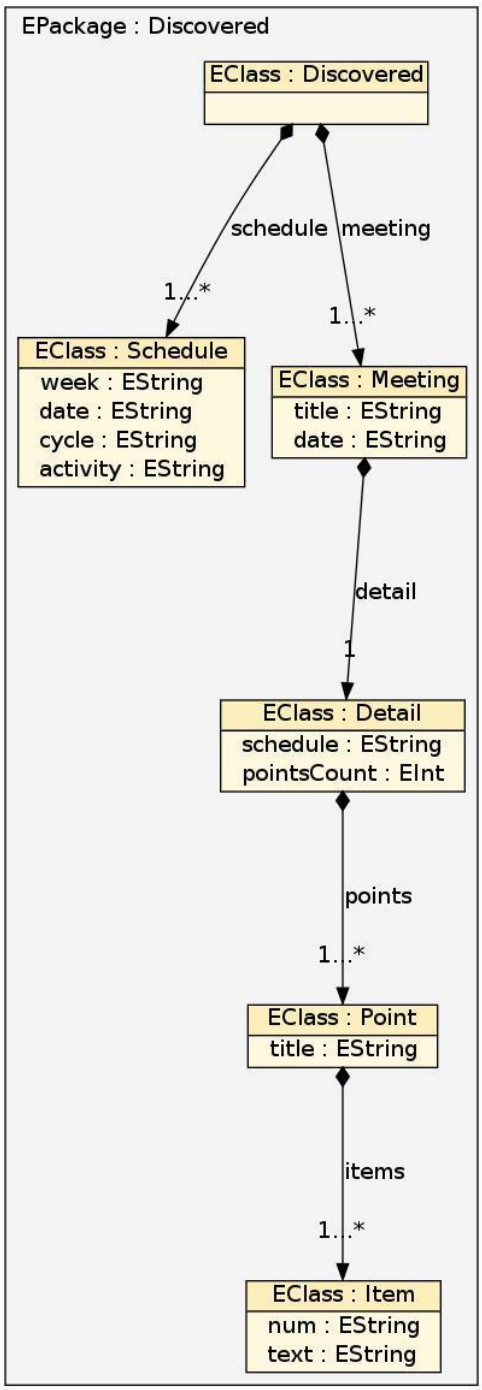
Na následujícím obrázku jsou znázorněny vztahy mezi tabulkami uvnitř databáze v podobě ERD diagramu. Každá zpráva *Item* musí mít určený právě jeden zdroj *Source* a zdroj může obsahovat libovolný počet zpráv. Označení PK1 znamená, že daný atribut je primární klíčem a FK určuje atribut, jenž slouží jako cizí klíč.



Obr. 11 ERD diagram databáze – notace Crow's foot

C Schéma JSON souboru

Na obrázku lze vidět UML diagram znázorňující specifické schéma JSON souboru, který nám poskytuje REST API a aplikace z něj načítá podstatné informace.



Obr. 12 UML diagram představující schéma JSON souboru

D Přiložené CD

Přiložené CD obsahuje:

- Instalační soubor aplikace
- Zdrojové kódy a XML soubory uživatelského rozhraní aplikace
- Zdrojové kódy webového REST API