VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

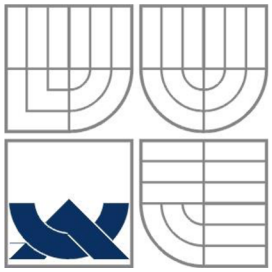
NÁSTROJ PRO SPRÁVU PROJEKTŮ S VYUŽITÍM TABLETŮ PLATFORMY ANDROID

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

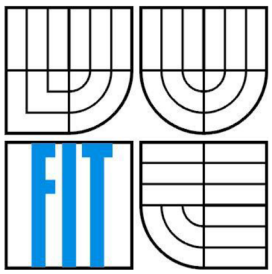
AUTOR PRÁCE
AUTHOR

Bc. JAN KOUDELKA

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

NÁSTROJ PRO SPRÁVU PROJEKTŮ S VYUŽITÍM TABLETŮ PLATFORMY ANDROID

PROJECT MANAGEMENT TOOL FOR ANDROID PLATFORM TABLETS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN KOUDELKA

VEDOUCÍ PRÁCE
SUPERVISOR

Mgr. PETR ŠKODA

BRNO 2014

Abstrakt

Tato práce se věnuje problematice řízení projektů a jejich podpory pomocí softwarových nástrojů. Zaměřuje se na analýzu existujících řešení a návrh vlastní mobilní aplikace pro platformu Android. Shrnuje základní principy tvorby mobilních aplikací pro Android OS. Navržená aplikace využívá vlastností cloudu pro snadnou synchronizaci dat mezi uživateli a pro jednoduché škálování.

Abstract

This thesis is dedicated to the issue of project management and its support using software tools. It is focused on analysis of existing solutions and designing a new mobile application for Android platform. It summarizes basic principles of making mobile applications for Android OS. Designed application uses some properties of cloud computing for scaling and synchronizing data easily.

Klíčová slova

Android, Cloud, PaaS, Google App Engine, tablet, mobilní zařízení, řízení projektů.

Keywords

Android, Cloud, PaaS, Google App Engine, tablet, mobile devices, project management.

Citace

KOUDELKA Jan: Nástroj pro správu projektů s využitím tabletů platformy Android, diplomová práce, Brno, FIT VUT v Brně, 2014

Nástroj pro správu projektů s využitím tabletů platformy Android

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením Mgr. Petra Škody. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Koudelka
28. května 2014

Poděkování

Touto cestou bych rád poděkoval vedoucímu práce Mgr. Petru Škodovi za ochotu při vedení práce, poskytnutí odborných rad, připomínek, konzultací a doporučené literatury. Dále bych chtěl poděkovat všem, kteří mě při psaní této práce jakkoliv podporovali.

Obsah

1 Úvod

Jak je z názvu této práce zřejmé, zaměřuje se na dvě oblasti, které v současné době zažívají obrovský nárůst. Především první z oblastí týkající se mobilních zařízení je dnes mezi dalšími technologiemi, které jsou v hledáčku široké veřejnosti, na předních místech a neustále se vyvíjí. Každý týden či dokonce den přichází na svět nové a nové mobilní technologie, kterým je třeba se přizpůsobovat. Zažíváme tak dobu, kdy mobilní zařízení pomalu nahrazují osobní počítače a notebooky. Například jen během prvního čtvrtletí roku 2013 vzrostl prodej dotykových tabletů celosvětově o cca 150 %. Druhou oblastí, na niž se práce soustřeďuje je problematika projektového managementu, neboli řízení projektů. Toto téma se v současnosti stává zájmem mnoha nově vznikajících firem, ale i těch existujících na trhu již několik let. Spolu s metodami procesního řízení bývá projektový management správnou cestou k úspěchu a prosperitě firmy.

Cílem této práce je seznámit se s principy tvorby mobilních aplikací na platformě Android, dále získat povědomí o možnostech použití cloudových technologií, zejména ve spojení s mobilními zařízeními. Úkolem je pak z dostupných řešení PaaS vybrat vhodného kandidáta pro implementaci rozhraní backendu, komunikujícího s mobilní aplikací na zařízeních Android. Hlavním cílem je na základě všech předchozích znalostí navrhnout aplikaci pro správu projektového portfolia.

Kapitola následující bezprostředně po úvodu se zabývá charakteristikou platformy Android z hlediska historie a současného vývoje. Dále popisuje její architekturu a principy tvorby aplikací, včetně klíčových aspektů aplikace a představení vývojového prostředí.

Kapitola číslo tři je stručným uvedením do problematiky projektového managementu. Poskytuje definice některých základních pojmů. V jednotlivých fázích popisuje celý životní cyklus projektu. V rámci této kapitoly jsou dále uvedeny používané nástroje pro řízení projektů. Na závěr pak následuje analýza několika existujících řešení softwarových produktů pro podporu projektového managementu.

Čtvrtá kapitola se zabývá obecnou charakteristikou cloudových technologií a samotnou definicí cloudu. Uvádí výhody a nevýhody plynoucí z použití cloudu. Následuje rozdělení těchto technologií podle různých kritérií, následované výběrem vhodného PaaS kandidáta pro další návrh.

Dále se pak v páté kapitole práce věnuje samotnému návrhu aplikace pro správu projektů. Tato část obsahuje specifikaci požadavků a řadu návrhových diagramů, včetně nastínění grafické podoby a návrhu loga aplikace.

Šestá kapitola popisuje realizaci aplikace dle návrhu z předchozí části práce. Uvádí implementační detaily klientské i serverové části aplikace. Předposlední částí je kapitola, věnující se testování implementované aplikace na reálných zařízeních s operačním systémem Android.

Závěr práce se zaměřuje na zhodnocení dosažených výsledků, rekapitulaci uskutečněných prací a nastínění dalšího možného vývoje navazujícího na tuto práci.

2 Platforma Android OS

Android je operační systém, který byl již od svého počátku určen a navržen pro velmi rychle rozvíjející se oblast komunikačních technologií, jakou mobilní zařízení bezpochyby jsou. Na rozdíl od svého největšího současného konkurenta, kterým je iOS od firmy Apple, je Android šířen naprosto volně pod licencí open-source[1]. Jinou konkurencí by mu mohl být momentálně jedině Windows Phone s rostoucím podílem na trhu, nicméně jeho současné rozšíření prozatím nijak neohrožuje vysoké procento podílu systému Android. Dnes je určen primárně pro mobilní telefony a jiná zařízení s dotykovými displeji, jako jsou dotykové tablety, multimediální centra, chytré telefony, televize, mini počítače a další.

2.1 Historie

Historie tohoto operačního systému sahá přibližně do roku 2003, kdy skupina čtyř lidí v čele s Andy Rubinem založila společnost Android, Inc [4]. Dalšími třemi spoluzakladateli byli Rich Miner, Nick Sears a Chris White. Mimochodem název Android vznikl právě ze zakladatelovy vášně k robotům (tzv. droidům) a z jeho přezdívky, kterou si shodou okolností vysloužil od svých kolegů za dob, kdy ještě pracoval u společnosti Apple. O dva roky později v roce 2005 firma přechází pod křídla společnosti Google, která za další dva roky nato oficiálně oznamuje vývoj nového systému Android OS, jenž by měl být plně šiřitelný pod licencí open-source [3].

Za necelý rok od tohoto okamžiku přichází Google s první verzí systému Android, který byl nasazen v mobilním telefonu od firmy HTC a spolu s ním byla uvolněna první verze SDK 1.0. Velmi rychle poté ji následovala vydání nových verzí, které byly (a dodnes jsou) pojmenovávány podle názvů různých zákusků a sladkostí, např. Donut, Honeycomb, Ice Cream Sandwich nebo nejnovější verze Android 4.4 – KitKat. Rozšíření jednotlivých verzí systému uvádí *Tabulka 1*. Verze Androidu starší než 2.2 nejsou uvedeny. Jejich rozšíření je v porovnání s ostatními minimální a s neustále vyvíjející se platformou tyto starší verze pomalu mizí. Tyto verze navíc nepodporují novou aplikaci GooglePlay, na základě které byla následující data měřena [1].

Během několik let získal operační systém Android mnoho příznivců a v průběhu svého působení se dostává z pozice vážného konkurenta společnosti Apple do vedoucího postavení na poli mobilních operačních systémů.

[1] Open-source je otevřený software, jehož zdrojové kódy jsou poskytovány za účelem volného užívání, úprav ve smyslu změny kódu a jeho dalšího šíření, za dodržení určitých podmínek licence.

Verze	Označení	API	Rozšíření
2.2	Froyo	8	1.0 %
2.3.3 – 2.3.7	Gingerbread	10	16.2 %
3.2	Honeycomb	13	0.1 %
4.0.3 – 4.0.4	Ice Cream Sandwich	15	13.4 %
4.1.x	Jelly Bean	16	33.5 %
4.2.x		17	18.8 %
4.3		18	8.5 %
4.4	KitKat	19	8.5 %

Tabulka 1. Přehled jednotlivých verzí systému Android a jejich rozšíření k datu 1. května 2014[2].

2.2 Současná situace

V době psaní této práce je nejnovější verzí na trhu Android 4.4 (KitKat). Tento operační systém podporuje řadu funkcí, které mají co nejefektivněji přispět ke snadnosti ovládání mobilních zařízení. Poskytuje řadu uživatelských gest pro ovládání dotykového displeje, včetně vícedotykových (multi-touch) gest[3]. Umožňuje používání široké škály widgetů a aplikací dostupných na Google Play Store. Podporuje multi-tasking, notifikace a vyspělé technologie, mezi které patří např. hlasové zadávání a ovládání aplikací. Mimo to jsou samozřejmostí služby jako Google Maps, nástroje pro zpracování fotografií a videa a další řada nástrojů, které jsou v dnešní době neodmyslitelnou součástí mobilních zařízení [2].

Rok 2013 je navíc přelomovým okamžikem v historii OS Android. Zakladatel a dlouholetý vedoucí vývoje tohoto operačního systému Andy Rubin odchází deset let od založení projektu z vedoucí pozice a na jeho místo nastupuje Sundar Pichai – produktový manažer nejdůležitějších služeb společnosti Google, ze kterých můžeme jmenovat např. webový prohlížeč Google Chrome[4]. Andy Rubin však z Googlu neodchází, pouze přechází k práci na tajném projektu firmy, kde bude moci své zkušenosti lépe uplatnit.

Abychom si udělali představu o tom, jak si Android na poli operačních systémů pro mobilní zařízení stojí, uveďme alespoň některá aktuální čísla. Zřejmě nejvíce vypovídajícím údajem, který

[2] Zdroj: http://developer.android.com/about/dashboards/index.html.

[3] Multi-touch je technologie dotykových zařízení umožňující rozeznání dvou a více kontaktních bodů, ve kterých se prsty uživatele stýkají s povrchem dotykového displeje.

[4] Zdroj: http://mobil.idnes.cz/andy-rubin-odchod-od-androidu-dji-/mob_tech.aspx?c=A130321_174310_mob_tech_ham.

člověku dokáže přiblížit situaci platformy Android v celosvětovém měřítku je číslo, které nedávno uveřejnil nynější vedoucí vývoje Sundar Pichai. Na svém Google+ profilu zveřejnil článek, ve kterém uvádí, že ke dni 3. září 2013 dosáhli počtu jedné miliardy zařízení s operačním systémem Android.

Během několika posledních let podíl Androidu mnohonásobně narostl a v dnešní době je již jedničkou na trhu. Podle analytické společnosti Kantar Worldpanel Comtech činí podíl Androidu na evropském trhu úctyhodných 70,4 %, čímž naprosto dominuje ostatním zástupcům z řad mobilních operačních systémů. Daleko za ním pak následuje se 17,8 % Apple se svým iOS a Windows s pouhými 6,8 %. V USA je poměr dvou hlavních konkurentů poněkud vyrovnanější. Stále však na prvním místě stojí Android s 52% podílem na trhu a na druhém místě iOS s 41,9 procenty. Na americkém trhu však není nárůst platformy Android tak rychlý jako v Evropě[5]. Zajímavě se však dnes vyvíjí podíl operačních systémů Windows, který vykazuje největší růst (v Evropě), ze všech uvedených - viz *Tabulka 2*.

Evropa - TOP 5	3 m/e Aug 2012	3 m/e Aug 2013	% pt. Change
Android	68.8	70.1	1.3
BlackBerry	5.8	2.4	-3.4
iOS	14.1	16.1	2.0
Windows	5.1	9.2	4.2
Other	6.1	2.1	-4.0

Tabulka 2. Aktuální vývoj OS s největším podílem na trhu[6].

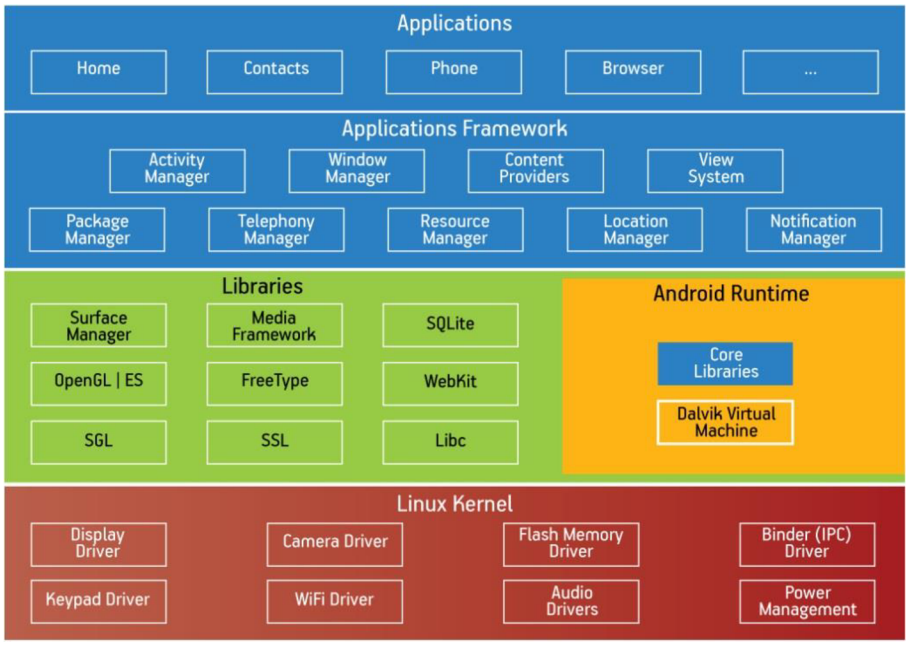
2.3 Architektura

Jak je znázorněno na *Obrázek 1*, rozděluje se architektura operačního systému Android na několik částí. Dle uvedeného modelu se konkrétně jedná o 5 klíčových vrstev, na nichž je tento systém postaven.

Na nejnižší úrovni se nachází jádro, pod kterým se neskrývá nic jiného než vhodně upravené a pro účely Androidu přizpůsobené jádro open-source operačního systému Linux. Toto jádro obsahuje základní ovladače a řadiče pro obsluhu hardwarového vybavení zařízení, na němž je Android nasazen. Mimo jiné jsou to hlavně ovladače pro správu napájení a paměti, dále pak pro řízení a přístup k různým funkcím zařízení jako např. fotoaparát (kamera), klávesnice, WiFi modul, zobrazovací zařízení a další. Na vyšší úrovni se nachází knihovny vytvořené v programovacím jazyce C/C++ [6], které slouží k obsluze funkcí na nižší úrovni linuxového jádra. Tyto knihovny poskytují funkce pro zpracování obrazu, přehrávání zvuků, či videa vyšším vrstvám architektury. Na obrázku je dále na stejné úrovni žlutou barvou vyznačen virtuální stroj Dalvik Virtual Machine. Jedná se

[5] Zdroj: http://www.svetandroida.cz/android-se-70-podilem-valcuje-evropu-ios-se-krci-v-koutku-201307.
[6] Zdroj: http://www.kantarworldpanel.com/dwl.php?sn=news_downloads&id=304.

o virtuální stroj podobný tomu z prostředí jazyka Java. Na rozdíl od JVM neobsahuje knihovny AWT, Swing a další a je plně přizpůsoben potřebám systému Android. O další úroveň výše se již dostáváme k velmi důležité vrstvě, kterou je aplikační framework pro podporu aplikací na systému Android. Pro programátory a vývojáře se jedná asi o nejdůležitější část této architektury. Obsahuje totiž služby nezbytné při vývoji aplikací, určené k obsluze elementárních stavebních bloků aplikace. Těmto základním prvkům, ze kterých jsou aplikace v systému Android tvořeny, se věnuje následující kapitola 2.4.



Obrázek 1. Schéma architektury operačního systému Android[7].

Co se typu architektury týče, jde v tomto případě o koncepci typu MVC (Model-View-Controller) [5]. Začneme-li u *Controlleru*, je jeho hlavní otázkou „co se má provádět“. Controller tedy řídí akce vyvolané různými aktivitami, např. uživatelem při stisknutí tlačítka. Jako reakci na tuto událost pak volá funkce dalšího na řadě, kterým je *Model*. Model zde představuje konkrétní funkcionalitu, tedy výkonnou část aplikace a říká „jak se to má udělat“. Obsahuje implementaci obslužných funkcí, případně může obsahovat různé instance databází, apod. Při vykonání určité operace je pak ve většině případů nutné vyvolat překreslení obrazovky pro zobrazení aktualizovaných výstupů. O to a o další problémy týkající se v první řadě grafické prezentace aplikace se stará modul architektury zvaný *View*.

[7] Zdroj: http://www.techdesignforums.com/practice/technique/android-beyond-the-handset/.

2.4 Základní komponenty aplikací Android

V této kapitole se krátce zmíníme o základních prvcích aplikací pro Android. Názvy jsou záměrně uváděny v originálním znění (angličtina), protože některé z jejich českých ekvivalentů nemusí zcela přesně vystihovat danou problematiku. Dále v textu pak mohou být použity počeštěné názvy pro lepší čitelnost v rámci celého kontextu. Následující podkapitoly částečně čerpají ze zdrojů [5][7].

2.4.1 Activity

Z toho nejjednoduššího pohledu můžeme aktivitu chápat jako jednu konkrétní obrazovku naší aplikace. Tato obrazovka obsahuje různé prvky grafického uživatelského rozhraní (GUI), komponované do tzv. *View* (pohledů), prostřednictvím kterých mohou být vyvolány určité akce. Mezi tyto akce kromě jiných patří i vyvolání jiné aktivity, což v praxi většinou znamená přechod na jinou obrazovku aplikace. Tyto přechody mezi jednotlivými obrazovkami jsou řízené pomocí tzv. *Intent*, jež popisuje následující podkapitola. O spouštění, řízení a ukončování aktivit se stará tzv. *Activity Manager*, který se nachází ve vrstvě aplikačního frameworku, uvedeného v předchozí kapitole na *Obrázek 1*. Z praktického hlediska jsou aktivity vytvářeny rozšířením (v jazyce Java použitím klíčového slova `extends`) třídy `android.app.Activity`. Je tedy zřejmé, že většina aplikací, pokud vyloučíme ty nejjednodušší, se bude skládat z několika aktivit. Aktivity jedné aplikace mohou být rovněž spouštěny z jiné, čímž je umožněno na platformě Android vyvíjet spolupracující aplikace, popřípadě aplikace, využívající jiných služeb než svých.

2.4.2 Intent

Jak již bylo řečeno v předchozí podkapitole, přechod mezi dvěma aktivitami není v režii těchto aktivit, ale je použito jiných prvků systému Android a tím jsou právě zmíněné Intenty. Kromě aktivit mají intenty na starosti také aktivaci *Services* a *Broadcast receivers*, prostřednictvím asynchronních volání. Intenty tedy propojují komponenty aplikace s jinými komponentami téže, či jiné aplikace. Pro aktivity a služby (services) definují intenty akce, které se mají provést (zobrazení jiné aktivity, zaslání zprávy) a mohou také definovat URI na data určená ke zpracování. Typickým příkladem může být výběr jednoho záznamu ze seznamu kontaktů, který bude dále v aplikaci nějakým způsobem použit. Aktivita vyžadující záznam ze seznamu kontaktů tedy vytvoří *intent* s požadavkem na aktivitu umožňující uživateli výběr kontaktu (pro tyto a jiné základní operace je možné využít základních systémových aktivit). Následuje uživatelská interakce, kdy uživatel vybere daný záznam. Nakonec je opět prostřednictvím intentu navráceno URI na konkrétní uživatelem vybraný záznam, který může být dále zpracován v původní aktivitě [7].

2.4.3 Service

Služby (services) lze vytvářet rozšířením třídy `android.app.Service`. Jedná se většinou o dlouho trvající úlohy běžící na pozadí. Jejich aktivita tedy nemusí být přímo viditelná. Může se jednat například o přehrávání hudby při současném prohlížení webových stránek, apod. Lze je také využívat pro sdílení funkcí aplikací prostřednictvím dlouhodobých spojení, přičemž tato služba běží v samostatném vlákně a nenarušuje tedy provádění hlavního vlákna programu. Aplikace tak může například asynchronně zpracovávat data zasílaná z jiných aplikací, aniž by přitom blokovala provádění aktuální aktivity aplikace.

2.4.4 Content provider

Content provider je komponenta, která umožňuje efektivně spravovat data aplikace. Ať už se jedná o data, která jsou sdílena nebo o soukromá data aplikace, existují různé možnosti jejich uložení. Mohou být uložena v databázi, v souborovém systému nebo např. na webu. Podstatné je, že prostřednictvím *Content Provider* je možné k těmto datům přistupovat, vytvářet nad nimi dotazy (v případě SQL), měnit obsah uložených dat a samozřejmě číst a zapisovat. Základními čtyřmi operacemi jsou *insert, query, update, delete*, případně určité modifikace a rozšíření nad těmito základními metodami.

2.4.5 Broadcast receiver

V podstatě se jedná o komponentu, sloužící k zachycení určitých událostí v systému. Mohou to být události vyvolané aplikacemi nebo například varování systému o nízké úrovni nabití baterie. *Broadcast Receiver* podobně jako předešlé *Activity* a *Service* využívá intentů. Na rozdíl od aktivit však neposkytuje žádné uživatelské rozhraní. Jedná se spíše o nástroj, který zprostředkovává informace o událostech jiným komponentám, které mohou na tyto události patřičným způsobem reagovat. Jedinou viditelnou interakcí pro uživatele může být vytvoření oznámení v oblasti notifikační lišty.

2.4.6 Resources

Především v oblasti mobilních zařízení a multimédií není většina aplikací složena pouze ze zdrojových kódů. Minimálně jsou to základní grafické prvky, které obsahuje každá aplikace, kam můžeme zařadit například ikony, loga, obrázky tlačítek, různé oddělovače, apod. Mimo to se však velká část současných aplikací neobejde ani bez různých obrázků na pozadí, dekorativních grafických prvků, které dokreslují uživatelské rozhraní aplikace, či obrázků specifických pro konkrétní zaměření aplikace.

Nejsou to však pouze obrázky, které patří do této kategorie zdrojů. Jako zdroje jsou v Androidu chápány například i XML soubory popisující vizuální vzhled a styl aplikace, definovaný pomocí tzv. View (pohledů). Všechny tyto zdroje se nacházejí v podadresáři `res` a jsou rozděleny do následující adresářové struktury[8]:

`layout`

V tomto adresáři jsou umístěny všechny XML soubory obsahující popis struktury aplikace z hlediska vizuálního vzhledu. Ke každé aktivitě poskytující určité GUI se zde nachází odpovídající soubor s definicí layoutu.

`drawable`

Kategorie *drawable* je ve skutečnosti dělena do několika podsložek. Jsou to většinou tyto čtyři: `drawable-ldpi`, `drawable-mdpi`, `drawable-hdpi`, `drawable-xhdpi`. Toto rozdělení odpovídá různým typům a velikostem displeje zařízení, přičemž uvedené typy jsou čtyři nejrozšířenější. Tyto adresáře obsahují bitmapové obrázky, které by měly být pro správné zobrazení optimalizované pro daný typ displeje.

`values`

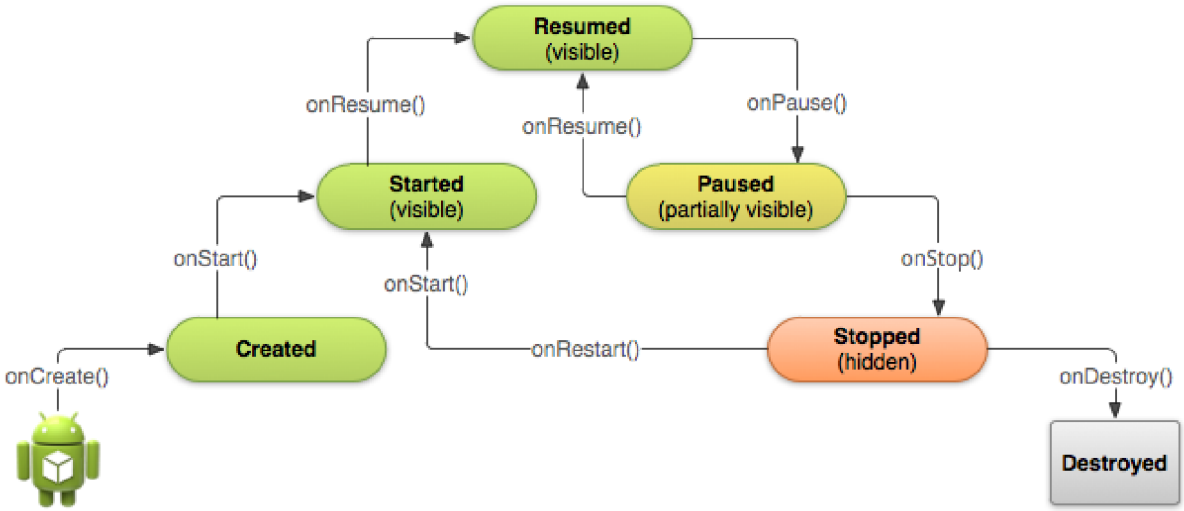
Adresář *values* obsahuje hodnoty, které jsou používány během provádění aplikace. Opět jsou uloženy ve formě XML souborů. Může se jednat např. o soubory definující rozměry komponent, styly nebo statické textové řetězce. Obzvláště poslední zmíněná možnost je skvělým nástrojem pro snadné vytváření vícejazyčných aplikací. Pokud jsou veškeré texty uloženy mimo aplikační kód, lze je snadno upravovat a převádět do jiných jazyků. Jednoduchá a přehledná struktura XML souboru navíc umožňuje vytvářet překlady těchto aplikací i osobám bez znalostí programovacího jazyka.

[8] Jedná se pouze o nejjednodušší možnou implementaci při vytváření jednoduché aplikace. Složitější aplikace mohou zdroje rozdělovat do více podadresářů, podle velikosti displeje, typu zařízení, či verze operačního systému.

2.5 Životní cyklus

Již bylo řečeno, že hlavními výkonnými prvky aplikací v androidu jsou aktivity. Tyto aktivity se během své doby života mohou nacházet v různých fázích životního cyklu, vyznačeného na *Obrázek 2*. Už na první pohled můžeme vidět zásadní rozdíl od jiných programovacích paradigmat, na která jsme nejspíše zvyklí. U těchto jiných paradigmat, ať už se jedná o programovací jazyky objektově-orientované nebo funkcionální, jsme byli zvyklí na způsob spuštění programu prostřednictvím hlavní metody, respektive funkce `main()`.

Jak ovšem vidíme na obrázku níže, v prostředí OS Android je situace poněkud odlišná a to dosti zásadním způsobem. Jednotlivé stavy, v nichž se aplikace může nacházet, tvoří vizuálně jakousi pyramidu, přičemž od svého spuštění prochází aplikace zleva postupně k vrcholu této pyramidy. Poté z vrcholu může postupovat pouze dále vpravo. Pokud aplikace běží na popředí, nachází se na vrcholu pyramidy. Dále vpravo se nacházejí stavy, které vedou k ukončení aplikace (viz následující text).



Obrázek 2. Schéma životního cyklu aktivity v Android OS[9].

Na výše uvedeném obrázku jsou kromě stavů (Created, Started, Resumed, Paused, Stopped, Destroyed) uvedeny také příslušné metody (na šipkách), které vyvolávají přechod mezi těmito stavy. Poté co uživatel stiskne ikonu aplikace, jsou velmi rychle po sobě volány metody `onCreate()`, `onStart()` a `onResume()`. Stavy *Created* a *Started* jsou tedy pouze přechodné a program v nich delší dobu nesetrvává [8]. Přechodem do *Resumed* se aplikace dostává do stavu, kdy běží na popředí a uživatel s ní může provádět dané interakce. Voláním `onPause()` přechází aktivita do stavu, kdy již není v interakci s uživatelem. V tomto stavu je například pouze částečně překryta jinou aplikací běžící na popředí. Dalším stupněm je stav *Stopped*, kdy dochází k přesunu aplikace do pozadí. Aktivita již

[9] Zdroj: http://developer.android.com/training/basics/activity-lifecycle/starting.html.

není viditelná, ale veškeré uživatelské proměnné a data jsou zachovány pro pozdější možnost obnovení aplikace prostřednictvím volání metody `onRestart()`, která vynutí znovuobnovení stavu *Started*. Druhou možností může být ukončení běhu aplikace voláním `onDestroy()`. Tato metoda je volána po předchozím volání metod `onPause()` a `onStop()`. Výjimku tvoří aktivity, které jsou vytvořeny pouze za účelem spuštění jiných aktivit a ihned poté zanikají. V takovém případě je vhodné využít metody `finish()` volané přímo z `onCreate()` metody dané aktivity. Tato metoda se již postará o vyvolání `onDestroy()` a o korektní ukončení aktivity [4][8].

S ohledem na náročnost dané aktivity je možné, že některé z uvedených metod nebude nutné vůbec implementovat. Je však přinejmenším vhodné je znát a vědět jak fungují, abychom vždy dosáhli těch nejlepších výsledků co se stability a výkonnosti aplikace týče. Správnou implementací všech potřebných metod dokážeme zajistit, že:

- Aplikace nehavaruje, dojde-li k přepnutí na jinou aplikaci, případně vlivem příchozího hovoru (u telefonů).
- Aplikace nespotřebovává zbytečně systémové zdroje, když ji aktivně nepoužíváme.
- Aplikace uchová uživatelská data (například postup ve hře), když ji vypneme a znovu spustíme.
- Aplikace uchová uživatelská data a nespadne v případě, že dojde k otočení displeje [8].

2.6 Vývojové prostředí

Jelikož se programování pro platformu Android dosti podobá klasickému programování v jazyce Java, ale samozřejmě i díky velké podpoře, rozšíření a licenci open-source, zvolili vývojáři tohoto systému jako oficiální nástroj pro vývoj aplikací prostředí Eclipse. Tento volně dostupný program je v dnešní době velmi používaný v různých odvětvích programovacích jazyků. Velmi často je pak spojován právě s vývojem aplikací v jazyce Java. Vývojáři systému Android vytvořili pro programátory speciální verzi tohoto nástroje upravenou pro potřeby vývoje aplikací na tomto systému. Balíček obsahující jádro nástroje Eclipse, včetně ADT pluginu a správce instalovaných SDK se nazývá ADT Bundle. Jeho aktuální verze je ke stažení na oficiálních stránkách Android Developers[10]. Kromě již uvedených součástí dále zahrnuje nejnovější verzi systému Android, řadu nástrojů pro podporu vývoje a distribuce aplikací a emulátor pro testování aplikací bez nutnosti vlastnictví reálného zařízení.

Jinou, relativně novou možností je použití aplikace Android Studio, jejíž první verze byla uvedena v roce 2013. Nyní je k dispozici ve verzi Android Studio v0.5.2. Tato aplikace je na rozdíl od Eclipse plně optimalizována a předurčena výhradně pro vývoj aplikací na platformě Android. Podobně jako Eclipse obsahuje širokou škálu nástrojů pro vývoj a ladění. Navíc má zabudovanou

[10] Web: http://developer.android.com/sdk/index.html.

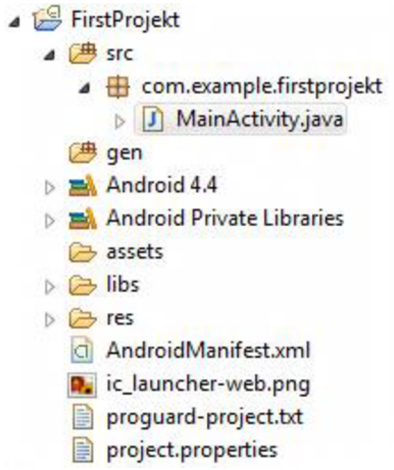
podporu pro vývoj aplikací na platformě Google Cloud s využitím Google Cloud Messaging. Nicméně se stále jedná o čerstvou aplikaci, která se postupně vyvíjí a není ještě zcela vyladěna. Především z tohoto důvodu bylo raději vybráno prostředí aplikace Eclipse.

Instalace a nastavení vývojového prostředí jsou krok za krokem popsány na uvedených webových stránkách, tedy se o nich nebudeme více rozepisovat. Spíše se podíváme na strukturu projektu v prostředí Eclipse a na tvorbu jednoduchého projektu.

Poté co je balíček ADT obsahující nástroj Eclipse nainstalován a nastaven dle výše zmíněného návodu na webu, je na řadě podívat se na samotné vývojové prostředí. Ti, kteří jsou s tímto nástrojem obeznámeni z dřívějších projektů, zřejmě nebudou nikterak překvapeni. Jedná se o zcela klasickou podobu programu Eclipse, který je doplněn přídavnými moduly pro platformu Android.

2.6.1 Vytvoření a struktura projektu

Založení nového projektu je naprosto jednoduché. V kontextovém menu programu Eclipse zvolíme: `File → New → Android Application Project`. V prvním okně zvolíme název aplikace, název projektu, případně jméno balíčku. Ve druhé části tohoto okna můžeme zvolit z nainstalovaných SDK minimální podporovanou verzi, dále pak verzi SDK, pro kterou se chystáme aplikaci vyvíjet a ve výběrovém poli *CompileWith* verzi SDK, na které bude aplikace kompilována. Přitom se doporučuje, aby možnosti *CompileWith* a *TargetSDK* obsahovaly vždy nejnovější verzi SDK. Kliknutím na `Next` můžeme přejít na další obrazovku. Ta obsahuje určité konfigurace projektu, které prozatím můžeme ponechat a přejít dále. Další obrazovka obsahuje konfiguraci spouštěcí ikony. Opět můžeme ponechat defaultní nastavení a přejít k dalšímu oknu. Následující okno vybízí k vytvoření aktivity a volby jejího typu. Zde označíme zaškrtávací políčko `Create Activity` a níže zvolíme typ `Blank Activity`. Následuje poslední obrazovka, kde opět můžeme nechat původní nastavení a stisknutím tlačítka `Finish` dokončit průvodce vytvořením projektu. Eclipse se následně postará o vytvoření projektu a příslušné struktury souborů, viz *Obrázek 3*.



Obrázek 3. Struktura projektu v prostředí Eclipse.

Na uvedeném obrázku je vidět, že byl vytvořen balíček `com.example.firstprojekt` a v něm jediný soubor `MainActivity.java` představující prázdnou aktivitu vytvořenou pomocí zaškrtávacího políčka v průvodci vytvořením projektu. Tento soubor obsahuje pouze dvě metody. První z nich je metoda `onCreate()`, nacházející se na prvním stupni pomyslné pyramidy dříve uvedeného životního cyklu aktivity. Druhou metodou je `onCreateOptionsMenu()`, která slouží pro přidávání prvků do tzv. action bar. V tomto momentě aplikace nedělá nic zvláštního, ale už nyní je možné ji spustit. Výsledkem je aplikace obsahující lištu a prázdnou obrazovku, která ovšem nic jiného nedělá.

Co se týká dalších součástí projektu, adresář `gen` obsahuje vygenerované soubory, které jsou vytvořeny při kompilaci. Do těchto souborů by uživatel neměl zasahovat. Dále následuje zvolené SDK (zde nejnovější Android 4.4 - KitKat), přiložené ve formě JAR archivu. Poté mohou v závislosti na zvoleném minimálním SDK následovat knihovny pro podporu starších verzí. Adresář `libs` je určen pro další externí knihovny, které může aplikace využívat. Struktuře a účelu adresáře `res` se již věnovala kapitola 2.4.6. Připomeňme pouze, že se jedná o složku obsahující zdroje, jako jsou obrázky nebo XML soubory obsahující definice struktury a stylů.

Nakonec se zmiňme ještě o neméně důležitém souboru `AndroidManifest.xml`. Tento soubor je v projektu nepostradatelný. Obsahuje důležité informace o projektu, jako jsou např. název, podporované SDK, minimální SDK, název výchozího balíčku, název hlavní spouštěcí aktivity, apod. Dále už následují jen soubory nastavení Eclipse, které se vztahují spíše k uživatelským nastavením vývojového prostředí a nemají tolik vliv na samotný projekt. Snad jen ještě nezapomeňme na ikonu nacházející se v hlavním adresáři projektu `ic_launcher-web.png`, která je určena pro zobrazení při distribuci aplikace na Google Play Store.

Tolik k samotnému úvodu o platformě Android a vývojovém prostředí pro tvorbu aplikací na tomto systému. Více k praktické části bude uvedeno dále v kapitole týkající se implementace navrhované aplikace, kde budou jednotlivé klíčové části detailněji rozebrány.

3 Projektový management

Přestože projekty jako takové řeší lidé již odpradávna, o projektovém managementu – neboli řízení projektů – můžeme říci, že se jedná o relativně mladou disciplínu, která se v dnešní době neustále rozrůstá, zdokonaluje a je zřejmé, že tomu tak bude i v dalších letech. Právě v současném moderním světě musí firmy, či podnikatelé pružně a svižně reagovat na rychle se měnící podmínky na trhu. Každodenně na tento trh vstupuje nová konkurence a firmy si tak musí neustále hlídat svoje postavení a strategickou výhodu pro udržení konkurenceschopnosti vůči ostatním subjektům. Kromě jiných podpůrných procesů jim k tomu dopomáhá právě projektový management.

Z názvu je jasné, že projektový management se týká především projektů. Tím, jak konkrétně je projekt definován a co všechno obnáší, se budeme zabývat v následující kapitole. Prozatím nám postačí základní představa, že se jedná o určitou časově ohraničenou posloupnost činností, ve kterých hraje roli několik zainteresovaných stran. Ve většině případů jsou navíc spotřebovávány určité zdroje. Typy zdrojů a ostatní atributy projektu budou taktéž zmíněny v následujících podkapitolách.

Nezávisle na velikosti a rozsahu řešeného projektu se projektový management snaží zodpovědět následující otázky [12]:

- Jaký problém má být řešen?
- Jakým způsobem bude problém řešen?
- Kdy je dosaženo cíle projektu?
- Jaký je výsledek projektu?

Každý projekt má určité faktory, které ho pozitivním či negativním způsobem ovlivňují. Mezi hlavní omezující podmínky dnes u většiny projektů patří zdroje, finanční prostředky a v neposlední řadě především čas [10]. Kvalita výsledného produktu, který realizace projektu přináší, pak v kombinaci s časem a finančními náklady tvoří tzv. projektový trojimperativ. Tento název je odvozen od závislosti těchto tří hodnot, ze kterých se skládá. Říká, že hodnota třetí veličiny je závislá na hodnotách ostatních dvou. A naopak pokud se jedna z veličin změní, automaticky musí dojít ke změně hodnot dvou zbývajících [12]. Model projektového trojimperativu je uveden na *Obrázek 4*.

Obrázek 4. Model projektového trojimperativu – kvalita, cena, čas [12].

3.1 Projekt

Existuje mnoho různých vysvětlení pojmu projekt, z nichž některé se liší více, avšak mnoho z nich si je velmi blízkých. Uveďme si tedy několik příkladů z různých zdrojů. Jedna z obecně známých definic projektu říká, že se jedná o časově ohraničenou posloupnost činností s jasně stanoveným začátkem a koncem, které s využitím zdrojů přeměňují vstupy na výstupy. Přitom je v průběhu projektu ke vstupům vytvářena určitá přidaná hodnota. Další z definic pochází od uznávaného amerického profesora Harolda Kerznera, který je odborníkem v oblasti projektového managementu, strategického řízení a mnoha dalších přilehlých odvětví. Poslední formulace je citací z knihy PMI (Project Management Institute).

Projekt je jedinečný sled aktivit a úkolů, který má:

- dán specifický cíl, který má být jeho realizací splněn,
- definováno datum začátku a konce uskutečnění,
- stanoven rámec pro čerpání zdrojů, potřebných pro jeho realizaci [13].

Projekt je dočasné úsilí vynaložené na vytvoření unikátního produktu, služby nebo určitého výsledku [14].

3.1.1 Klíčové aspekty projektu

Ať již dáme zapravdu kterékoliv z předešlých definic, všechny se setkávají v klíčových aspektech projektu, na které se nyní zaměříme.

Cíl projektu

Projekt může mít ve svém výsledku jeden či několik dílčích podcílů. Nicméně je důležité, aby každý z těchto cílů splňoval jisté podmínky. S ohledem na tyto podmínky se můžeme setkat se zkratkou SMART, která se v této oblasti používá. Jedná se o zkratku z anglických slov udávajících požadované vlastnosti projektového cíle, kterými jsou:

- **S**pecific (specifikované)
- **M**easurable (měřitelné)
- **A**chievable (dosažitelné)
- **R**ealistic (realistické)
- **T**imed (termínované)

Zdroje projektu

Za zdroje můžeme v projektu považovat jak materiální věci, tak lidské zdroje, které jsou pod kontrolou manažera. Ten prostřednictvím manažerských technik a nástrojů plánuje využití těchto zdrojů tak, aby byly co nejefektivněji využity za účelem vytváření výstupů, které jsou naplněním cíle projektu [9]. Zdroje dále můžeme rozdělovat na lidské, finanční, informační, hmotné a technologické. V praxi znamenají nadbytečné zdroje plýtvání finančními prostředky, což může vést k přetěžování dostupných zdrojů (především lidských) [11]. Z tohoto důvodu je důkladné plánování zdrojů a jejich přidělování důležitou součástí řízení projektů.

Vstupy a výstupy

Vstupy projektu jsou před jeho započetím dodány od zadavatele projektu. Většinou se jedná o neformální specifikaci požadovaného produktu, na základě které je později v úvodních fázích projektu vytvořena formální specifikace. Na druhé straně pak stojí výstupy, které uspokojují potřeby zadavatele a měly by být naplněním jím požadovaných cílů.

Zainteresované strany

Hlavními dvěma stranami, které jsou do projektu přímo zapojeny, jsou dodavatel a zákazník. Zákazník je osoba, či instituce, pro kterou je daný projekt vypracován. Dodavatel je opět osoba nebo firma, která na základě požadavků zákazníka a na základě vlastnictví zdrojů a výrobních prostředků provede daný projekt [9][14]. Chápeme-li dodavatele jako firmu, pak zajisté jako zainteresovanou stranu bereme v potaz i její zaměstnance, kteří se na daném projektu jakýmkoliv způsobem podílejí. Posledním významným subjektem, nikoliv však co se důležitosti týče, může být sponzor. Je to osoba, která je manažerem zákazníka projektu a má dostatečné pravomoce při rozhodování o rozpočtu, předmětu projektu a jiných zásadních věcech [9].

3.1.2 Fáze životního cyklu projektu

Během své existence se projekt nachází v různých stádiích vývoje, jejichž posloupnost označujeme pojmem životní cyklus projektu. Tyto jednotlivé části jsou pak nazývány fázemi projektu. Opět existuje řada různých interpretací, které se liší v uváděných termínech. Podle IPMA (International Project Management Association) [9] a dalších zdrojů [10][11] však lze životní cyklus projektu rozdělit do následujících pěti fází: zahájení, plánování, řízení (provedení), kontrola a ukončení. Na obrázku v *Příloha A* je znázorněna struktura projektu, rozdělená z jiného pohledu abstrakce. A sice na fázi předprojektovou, projektovou (zahrnující dříve zmíněných pět fází projektu) a poprojektovou. U jednotlivých fází jsou dále uvedeny klíčové úkoly vztahující se k aktuální fázi projektu. Dle IPMA® je fáze projektu definována jako skupina logicky souvisejících činností z pohledu řízení projektu. V celkové posloupnosti všech činností je fáze jasně vymezeným časovým úsekem, který je zřetelně oddělen od ostatních takových úseků [10]. V základu tedy každá fáze musí definovat úkoly, které se mají provádět s ohledem na konkrétní fázi projektu. Dále jsou to výstupy, jež fáze produkuje, a zdroje, které se aktivit v jednotlivých fázích účastní. Následující odstavce čerpají ze zdrojů [9,10,11,13].

Zahájení

Poté co bylo rozhodnuto o realizaci projektu, je nutné jej patřičným způsobem zahájit. Měla by být vytvořena zakládací listina projektu, obsahující všechny důležité technologické a organizační parametry. V této fázi by měly být rovněž stanoveny projektové týmy, určeny komunikační kanály a určeny osoby odpovědné za průběh jednotlivých fází.

Plánování

V tento moment jsou již stanoveny týmy pracovníků. Nyní je zapotřebí určit všechny činnosti v projektu. K tomu se využívá tzv. WBS (work breakdown structure) techniky, která pomáhá rozdělit jednotlivé cíle projektu do dílčích částí, až na úroveň jednoduchých činností. Na základě těchto činností jsou pak kladeny požadavky na stanovení rolí a jim odpovídajících odpovědností. Role jsou pak mapovány na zdroje. Přiřazení rolí a jim příslušejících odpovědností je zachyceno pomocí nástroje RACI, respektive RASCI[11] matice. V této fázi je stanoven základní harmonogram projektu nazývaný *baseline*[10].

Řízení

Na počátku této fáze je většinou pořádán tzv. kick-off meeting, kdy se sejdou zástupci všech zainteresovaných stran. Probíhá diskuse a obeznámení všech stran s plánem projektu, případně jsou provedeny změny plánu. Poté jsou všichni účastníci projektu informováni o tom, že realizace projektu

[11] RASCI = anglická zkratka pro různé stupně odpovědnosti; R=responsible, A=accountable, S=support, C=consulted, I=informed.

byla zahájena. Práce pak probíhají dle stanoveného plánu a jsou koordinovány odpovědnými osobami.

Kontrola

Během všech fází projektu je zapotřebí provádět měření metrik, kontrolovat plnění plánu a dodržování časových intervalů. Na základě těchto údajů pak vzniká ke konci projektu zpráva o celkovém průběhu projektu. Mimo to probíhá kontinuálně v rámci celého životního cyklu projektu analýza rizik, které mohou být potenciální příčinou vzniku škod. Dokument identifikující možná rizika a obsahující protiriziková opatření by proto měl být součástí prvních fází projektu. Důležitou částí fáze kontroly je také reakce na požadované změny v plánu projektu, kdy je zapotřebí zahájit změnové řízení, jehož výsledkem je upravený projektový plán, přerozdělení zdrojů a kalkulace nových nákladů na realizaci změn.

Ukončení

Přestože se jedná o poslední fázi, kdy je projekt téměř u konce, zahrnuje tato část životního cyklu řadu úkolů a událostí, které musí být splněny. Jedním z nejdůležitějších úkonů je příprava prostředí pro nasazení projektu. Tato část se týká především projektů z oblasti informačních technologií a v ostatních oblastech tedy nemusí být tak výrazná nebo nemusí být vůbec přítomna. Všechny typy projektů však mají společný určitý způsob akceptačního testování. Při tomto testování dochází ke konfrontaci zákazníka s hotovým produktem, přičemž je zkoumáno, zda produkt splňuje zákazníkovy požadavky. Na základě tohoto testování jsou vytvořeny protokoly, které jsou vystaveny podpisu zákazníka a slouží jako doklad o dodržení podmínek smlouvy z hlediska výsledného produktu.

Jelikož během projektu téměř vždy dochází k různým zdržením, chybám, či odchylkám, je nutné objektivně zhodnotit práci všech zapojených zaměstnanců a vyvodit příslušná opatření pro to, aby se již v budoucnu tyto chyby neopakovaly.

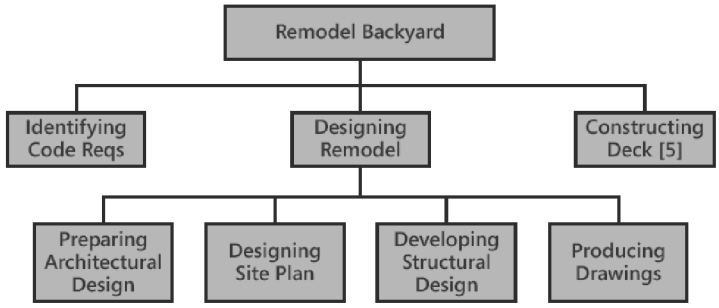
Poslední částí této fáze je samozřejmě předání cílového produktu zákazníkovi. V případě informačního zboží, či jiných IT produktů pak může následovat např. školení zaměstnanců zákazníka, instalace produktu ve firemním prostředí klienta, apod.

3.2 Nástroje projektového managementu

Pro podporu řízení projektů byla vynalezena široká škála podpůrných nástrojů a metodik, více či méně známých. Existují však ověřené postupy a techniky, které se v průběhu let staly jakýmsi standardem v projektovém managementu. Některé z níže uvedených technik, či nástrojů již byly zmíněny v předchozí podkapitole. Nyní budou blíže specifikovány z pohledu funkcí, které poskytují.

3.2.1 WBS – Work breakdown structure

Technika WBS nemá v češtině úplně přesný překlad, často se však využívá názvu hierarchická struktura činností. Tato metoda je určená pro rozklad komplexního problému řešeného projektu na dílčí činnosti. Může mít několik úrovní v závislosti na složitosti jednotlivých částí. Hierarchická struktura činností slouží pro stanovení harmonogramu prací v plánu projektu a je tvořena ve fázi plánování. Nejpřirozenější technikou použití hierarchické struktury je přístup shora dolů, neboli dekompozice [10], kdy se od hlavních výstupů přes dílčí výstupy dostáváme až k těm na nejnižší úrovni rozkladu. Opakem je přístup zdola nahoru, kdy jsou definovány všechny potřebné výstupy na nejnižší úrovni a z nich jsou pak následně skládány větší celky. Z pohledu hierarchického uspořádání lze tedy jednotlivé činnosti této struktury rozdělit na dva typy. Prvním typem jsou základní úkoly nacházející se na nejnižší úrovni, které symbolizují provedení konkrétní práce. Na druhé straně jsou to prvky, zahrnující provedení více úkolů, přičemž tyto úkoly mohou být na základní úrovni nebo se může jednat o souhrn komplexních činností na vyšší úrovni [12]. Vzniká tím logické uspořádání, které je velmi často znázorňováno pomocí tabulek nebo například formou stromové struktury, viz následující *Obrázek 5*.



Obrázek 5. Příklad hierarchické struktury činností (WBS) [12].

Jednotlivé dílčí činnosti přitom musí být v souladu se stanoveným časovým rámcem a v neposlední řadě s celkovým rozpočtem projektu. Alespoň pro klíčové úkoly se tedy doporučuje používat spolu s WBS například formulář „Pověření k provedení úkolu“, který blíže specifikuje zadání, rozsah a cíle úkolu. Pokud se jedná o významnou položku v rozpočtu, měla by být tato

informace jasně znatelná [11]. Úroveň dekompozice záleží na splnění několika faktorů, které musí platit pro činnosti na nejnižší úrovni. Tyto činnosti musí být měřitelné jak z hlediska průběhu, tak z hlediska výsledků, které poskytují. Dále je zapotřebí, aby měly rozumnou délku trvání z důvodu jednoduchého řízení přidělování zdrojů. Dalším kritériem je jasně stanovitelný začátek a konec a v neposlední řadě schopnost být realizovány bez čekání na výsledky jiných činností [12].

3.2.2 Ganttův diagram

Ganttův diagram je nástroj, který formou grafu umožňuje plánování a řízení projektu. Je pojmenován podle Henryho Laurence Gantta, který se zasloužil o jeho rozšíření. První zmínky o použití Ganttova diagramu pochází někdy z období první světové války, avšak většího rozšíření dosáhla tato metoda až s příchodem prvních počítačů [6]. Diagram se skládá z následujících tří prvků:

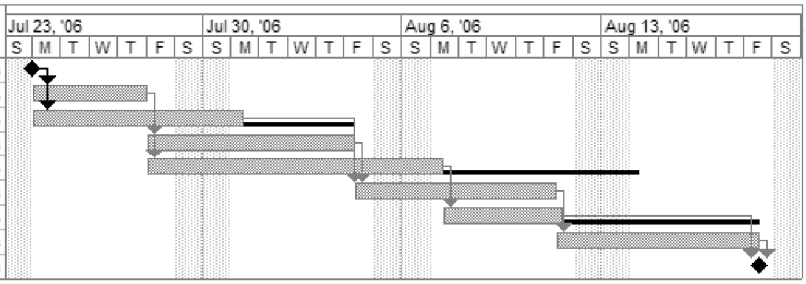
Milník Jedná se o časový okamžik v projektu, který má většinou nulovou délku trvání. V tomto okamžiku se měří postup v projektu. Probíhá kontrola plnění plánu projektu.

Úkol Jasně stanovená činnost v plánu projektu s pevně daným začátkem a koncem. Má obvykle krátkou dobu trvání a minimum interakcí s ostatními úkoly [9]. Jedná se o činnost, u které můžeme specifikovat její výsledky.

Souhrn Označuje množinu úkolů, které spolu určitým způsobem souvisí. Začátek souhrnu je přitom dán začátkem prvního úkolu v pořadí v čase a konec souhrnu určuje konec posledního úkolu v čase.

Ganttův diagram se skládá ze svislé osy, na které jsou uvedeny všechny úkoly v projektu, a z osy vodorovné, na níž se nachází časové měřítko (dny, týdny, měsíce, apod.). Tělo diagramu pak tvoří časové úseky značící dobu trvání jednotlivých dílčích úkolů a souhrnů s vyznačením definovaných milníků. Příklad krátkého výseku z takového diagramu ukazuje *Obrázek 6.*

ID	Task Name	Predecessors	Duration
1	Start		0 days
2	a	1	4 days
3	b	1	5.33 days
4	c	2	5.17 days
5	d	2	6.33 days
6	e	3,4	5.17 days
7	f	5	4.5 days
8	g	6	5.17 days
9	Finish	7,8	0 days



Obrázek 6. Ukázka časového plánování úkolů pomocí Ganttova diagramu[12].

[12] Zdroj: http://en.wikipedia.org/wiki/Gantt_chart.

3.2.3 Kritická cesta

Metoda kritické cesty je algoritmus, který pomocí orientovaného ohodnoceného grafu určuje nejdelší posloupnost činností od začátku do konce projektu. Činnostem ležícím na kritické cestě je třeba věnovat zvýšenou pozornost, neboť jejich prodloužení může mít za následek zpoždění celého projektu, což je většinou velmi nežádoucí. Pro kritickou cestu tedy platí, že:

- její celková délka určuje celkovou délku harmonogramu,
- zpoždění jakékoliv činnosti nacházející se na kritické cestě může vést k opoždění projektu,
- prodloužením činností mimo kritickou cestu se může stát kritickou cestou i jiná větev časového harmonogramu [9].

3.2.4 RASCI matice

Matice odpovědností (anglicky responsibility/accountability matrix) je nástroj pro stanovení kompetencí a odpovědností jednotlivých rolí, které se v projektu vyskytují. Asi nejznámějším typem takovéto matice je právě RACI, resp. RASCI matice. Význam této zkratky již byl dříve vysvětlen, uveďme tedy, co jednotlivá písmena v této matici znamenají a jak se používají.

Prvky WBS	**Manažer** Novák	**Člen týmu 1** Polák	**Člen týmu 2** Horák	**Člen týmu 3** Novotný	**Sub-dodavatel** Firma DATA	**Poradenský expert**	...
A...							
B...							
C...							
D nákup softwaru	A	R	R	–	I	C	
E...							

Obrázek 7. Ukázka RA(S)CI matice odpovědností [10].

Na *Obrázek 7* je vidět jakým způsobem je matice tvořena. Na levé straně tabulky jsou vypsány jednotlivé prvky WBS a v horizontální části záhlaví tabulky jsou uvedeni všichni zúčastnění pracovníci s definovanými rolemi. V těle tabulky jsou pak písmeny vyznačeny odpovědnosti pracovníků ve vztahu k jednotlivým činnostem dle následujícího významu písmen:

- **R**esponsible - osoba odpovědná za vykonání daného úkolu
- **A**ccountable - osoba, která zajišťuje správné plnění úkolu, ale nemusí přímo vykonávat danou činnost; deleguje zodpovědnost
- **S**upport - osoba zajišťující podporu při realizaci činnosti
- **C**onsulted - osoba poskytující rady a konzultaci při výkonu činnosti
- **I**nformed - osoba, která má být informována o průběhu a rozhodnutích

3.3 Existující softwarová řešení pro podporu řízení projektů

Mezi programy podporující řízení projektů můžeme v podstatě zařadit jakýkoliv software, který určitým způsobem usnadňuje některou z činností projektového managementu. Ve většině případů jsou to aplikace, které umožňují jednoduše vytvářet a používat nástroje uvedené v předchozí podkapitole a samozřejmě mnohé další.

Microsoft Project

Tento desktopový nástroj od společnosti Microsoft® je velmi dobrým pomocníkem pro řízení projektů. Nabízí možnosti vytvoření WBS, správy a plánování úkolů, zobrazení stavu projektu a řízení projektu pomocí Ganttova diagramu, správu zdrojů a další. Poskytuje různé výstupy, jako kalendáře, analýzy finančních toků, již zmíněné Ganttovy diagramy, vytížení zdrojů, atd. Samozřejmostí je také import/export z resp. do mnoha různých formátů, pro další možnosti zpracování. Tento nástroj existuje pouze v placené verzi a cena licence na jeho poslední verzi MS Project 2013 Professional se pohybuje okolo 30.000 Kč. Je vhodný pro řízení malých i velkých projektů s podporou týmové synchronizace prostřednictvím MS Share Point, případně MS Project Server.

ProjectManager.com

Sám název napovídá, že se jedná o online produkt pro podporu projektového managementu. Tento webový produkt nabízí tři programy, jejichž cena se pohybuje přibližně od 15 do 25 amerických dolarů za uživatele na měsíc. Tato aplikace umožňuje jednoduché řízení projektů a to velmi rychle, během pár kliknutí myší. Po registraci je navíc uživateli poskytnuta třicetidenní zkušební Trial verze. Aplikace zvládá opět veškeré standardní úkony projektového řízení, navíc přidává široké možnosti on-line týmové komunikace. Prostřednictvím různých zobrazení, lze sledovat vývoj projektu z několika různých pohledů. Navíc tato aplikace nabízí jednoduchou mobilní verzi, která umožňuje řídit projekty takřka odkudkoliv.

Project Libre

Jedná se o aplikaci napsanou v jazyce Java a šířenou jako open-source pod licencí CPAL. Project Libre je vyvíjen od roku 2012. Vychází však z předchozího projektu společnosti Serena Software, která od roku 2009 pracovala na projektu OpenProj, který je dodnes používán. Project Libre je do jisté míry kompatibilní s MS Project, a to především díky možnosti exportu do souboru typu MS Project XML.

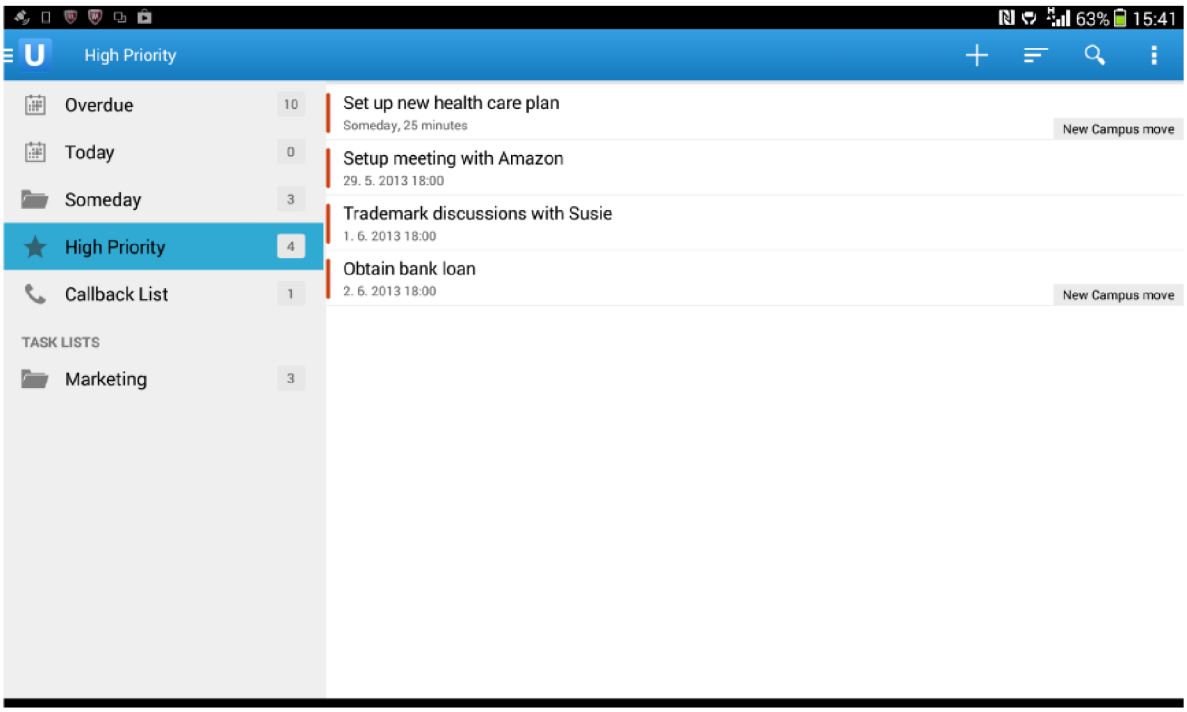
Následující dvě uvedená softwarová řešení jsou určena pro mobilní platformu, konkrétně pro operační systém Android OS.

Ap4Mobile

Aplikace Ap4Mobile je volně stažitelná z Google Play. Spíše než pro větší a formální projekty je určena především k organizaci menších osobních či školních projektů. Nabízí však velmi jednoduché a přitom uživatelsky příjemné grafické rozhraní. Umožňuje přehledně organizovat projekty, v rámci nichž lze spravovat tzv. to-do list, sdílené soubory, fotografie, zasílat pozvánky k připojení do projektu, apod. Spíše než pro obvyklé řízení použitím manažerských nástrojů, může být tato aplikace použita jako doplňující komunikační a koordinační prostředek.

Upvise Project

Opět se jedná o volně dostupnou aplikaci na platformě Android OS, která nabízí široké spektrum funkcí. Poskytuje podporu pro řízení projektů, definování jednotlivých činností a úkolů, správu a přiřazování zdrojů, zobrazení kalendáře, apod. Provázanost jednotlivých oblastí projektu je sice vhodně řešena, avšak v celkovém pojetí se ovládání aplikace jeví jako dosti chaotické a nepředvídatelné.



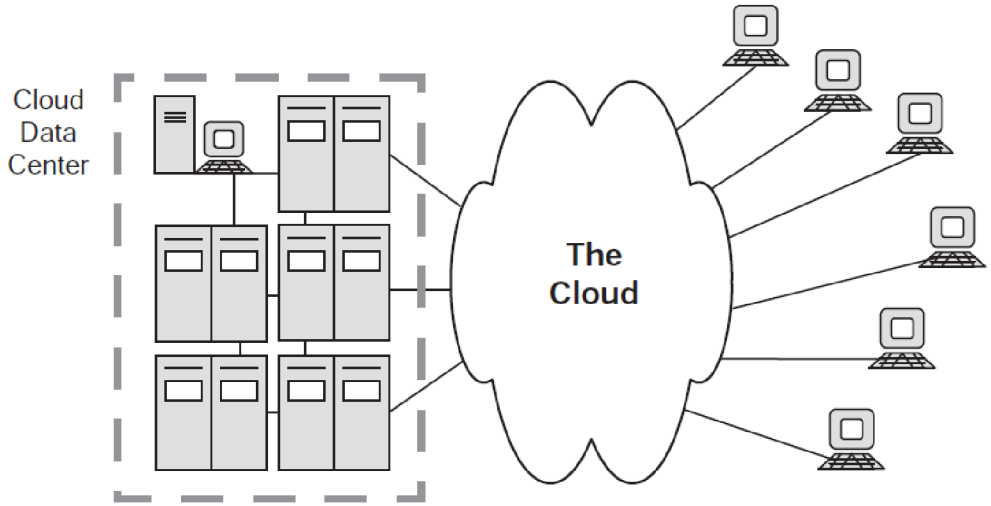
Obrázek 8. Ukázka obrazovky úkolů aplikace Upvise Project.

4 Cloud computing

Následující kapitola charakterizuje vlastnosti cloudových technologií. Věnuje se vysvětlení základních pojmů spojených s touto tématikou a analýzou různých typů cloud computingu. Dále shrnuje možné výhody při využití cloudu a na druhé straně i případné nevýhody. Na konci kapitoly jsou zmíněna některá existující řešení, z nichž jedna varianta bude vybrána pro další návrh a realizaci aplikace, jak plyne ze zadání této práce.

4.1 Obecný popis

Z pohledu vývoje informačních a komunikačních technologií se jedná o poměrně mladou a moderní technologii, která staví na robustní architektuře sítě Internet. Můžeme slyšet i o použití Cloudu v rámci soukromé sítě, avšak základním principem a myšlenkou, proč byla tato technologie vymyšlena, je její neomezená a jednoduchá dostupnost odkudkoliv z celého světa. Pokud bychom měli přesně definovat, o co se tedy jedná, je Cloud computing určitým modelem ke zpřístupnění sdíleného prostoru obsahujícího různé výpočetní a informační zdroje, vzdáleně prostřednictvím počítačové sítě. Mezi tyto zdroje pak můžeme řadit např. počítačové servery, datová úložiště, aplikace, služby, apod. [17]. Právě díky dnešnímu rozšíření mezinárodní počítačové sítě Internet je možné neustále tyto technologie vylepšovat a poskytovat jejich uživatelům stále lepší podmínky použití. V podstatě každý, kdo denně vykonává nějaké základní úkony s počítačem, jako např. posílání e-mailových zpráv, se tímto dostává do styku s cloudovými technologiemi. Nejedná se totiž o nic jiného než o to, že využíváme cizích služeb, poskytovaných prostřednictvím internetu, a to jakým způsobem jsou tyto služby řešeny, kolik si vyžadují výpočetních zdrojů a kolik místa na datovém úložišti je nám zcela skryto.



Obrázek 9. Jednoduché schéma Cloud computingu [19].

Základní myšlenka tedy tkví v tom, že aplikace a data jsou uloženy v počítačové infrastruktuře poskytovatele a uživatel k nim přistupuje přes rozhraní. Tímto rozhraním jsou většinou buď webové stránky, nebo aplikační rozhraní, která vzdáleně komunikují se zařízeními poskytovatele. Název si tento model poskytování služeb vysloužil ze stejného anglického slova *cloud* (mrak) právě díky své struktuře a způsobu, jakým jsou jednotlivé součásti infrastruktury koncentrovány do jednoho místa. Jednoduchý konceptuální model Cloud computingu je uveden výše na *Obrázek 9*.

4.2 Historie

Ohlédneme-li se zpátky do historie informačních technologií, můžeme sledovat, že počátky skutečného Cloud computingu přímo souvisejí s rozvojem World Wide Web (dále WWW) v 90. letech 20. století. Právě vznik WWW přinesl nové možnosti pro rozvoj podnikání a poskytování služeb typu internetového obchodování. V této době však můžeme skutečně mluvit pouze o počátcích Cloud computingu, který se pomalu začínal formovat s rozvíjejícím se internetem a celosvětovou sítí World Wide Web až do podoby v jaké ho známe dnes. Půjdeme-li však v historii ještě o několik let zpět, pak zjistíme, že již před touto érou existovaly technologie, které mohly být určitým náznakem počínajícího formování Cloud computingu. Mezi tyto technologie lze zařadit tzv. grid computing, adaptive computing, utility computing, či poskytování aplikačních služeb (application service providing) [15].

4.3 Výhody použití cloudu

Použití cloudu má velkou řadu výhod, z nichž za jednu z největších lze považovat širokou a snadnou škálovatelnost. V praxi to znamená, že nezáleží na tom, zda danou aplikaci využívá jeden, deset nebo stovky, či tisíce uživatelů. Záleží na poskytovateli služeb, za které platíme, aby zajistil dostatečný výkon pro aktuální potřeby našich aplikací. Škálovatelnost je tedy v podstatě měřítkem počtu uživatelů, kteří mohou k aplikaci přistupovat v jeden okamžik a efektivně ji využívat, aniž by byli jakkoliv omezeni výkonem [18]. V případě omezení to pak většinou znamená, že byl dosažen určitý limit škálovatelnosti. Je-li to možné, pak je v takovém případě žádoucí navýšit výkon přidáním výpočetních zdrojů.

Další nespornou výhodou, která již byla lehce nakousnuta, je skutečnost, že při využívání většiny služeb cloudu platí zákazník za to, co opravdu využije. Oproti pronajímání vlastního serveru, kdy často nevyužijeme plnou kapacitu prostředků, za něž platíme, se jedná o důležitou výhodu, která nám může ušetřit nemalé peníze.

Z popisu cloudových technologií je dále zřejmé, že odpadají veškeré starosti o údržbu výpočetních zařízení a provozu služeb. Vše je tak říkajíc pod jednou střechou a plně v režii poskytovatele cloudových služeb.

Z pohledu vývojářů aplikací je hlavní výhodou možnost odpoutat se od návrhů, které jsou založeny na dostupných výpočetních zdrojích. Mají tak možnost veškerou svoji pozornost soustředit na vytváření konkrétních produktů, bez nutnosti neustálé koordinace s hardwarovým zařízením.

Z jiného pohledu, který bude zajímat spíše manažery a vedoucí finančních oddělení, je výhodou také úspora nákladů plynoucí z faktu, že samotný pronájem služeb a zařízení je často mnohonásobně výhodnější, než pořízení vlastní infrastruktury. Je to především dáno tím, že zařízení na straně poskytovatele je sdíleno několika uživateli najednou. Čím větší základnu klientů poskytovatel má, tím výhodněji může stanovovat ceny.

4.4 Nevýhody použití cloudu

Z výše uvedených výhod, které Cloud computing nabízí, přímo plynou některá možná rizika a nevýhody. Tato rizika jsou však spíše otázkou menších poskytovatelů, kteří nemají stabilní postavení na trhu a u kterých tak mohou nastat potíže s dodržováním podmínek provozu.

Klíčovým aspektem a zřejmě nejčastějším bodem úrazu, co se týče dodržování smluvených podmínek, je špatná nebo naprosto chybějící definice tzv. SLA (service level agreement). Na základě takto špatně definovaných podmínek je pak velmi obtížné domáhat se u poskytovatele svých práv. Úzce s tím souvisí také dostupnost kvality služeb (QoS).

Další nevýhodou, která se může u těchto poskytovatelů vyskytovat je nedostatečná důvěryhodnost. Pokud chcete svá data svěřit do cizích rukou, pak je důvěryhodnost vůči danému subjektu na prvním místě. Rizika přímo plynoucí ze svěření dat do rukou nedůvěryhodných osob jsou například odcizení nebo ztráta dat.

Posledním rizikem spíše než nevýhodou je nestálost dodavatele služeb. Pokud existuje smlouva s poskytovatelem na dobu určitou, pak není jisté, že po uplynutí této doby bude možné smlouvu dále prodloužit. Může se stát, že je poskytovatel z finančních, či jiných důvodů nucen ukončit svoji činnost. Pak vyvstává problém s hledáním poskytovatele s podobnými podmínkami a nabídkami, případně nalezením jiného řešení, které může znamenat nadbytečné výdaje.

4.5 Rozdělení

Modely poskytovaných cloudových služeb můžeme dle různých hledisek dělit na několik kategorií. Dva nejobvyklejší způsoby rozdělení jsou uvedeny v následujících dvou podkapitolách.

4.5.1 Dle místa nasazení

Rozdělení podle místa nasazení říká, jakou cestou jsou služby cloudu zpřístupněny uživateli. Tato podkapitola částečně čerpá ze zdrojů [15][22].

Veřejný (Public cloud computing)

Veřejný cloud je často označován jako klasický. Jedná se o model cloud computingu, který je přístupný široké veřejnosti. Přitom všem uživatelům většinou nabízí stejné funkce, které mohou být odstupňovány prostřednictvím různých cenových balíčků.

Soukromý (Private cloud computing)

Soukromý cloud je provozován v domácím prostředí, či v rámci jedné firmy. Slouží výhradně pro účely dané firmy nebo osoby. V domácnostech bývá nejčastěji ve formě multimediálního centra.

Hybridní (Hybrid cloud computing)

Tento typ cloudu v sobě spojuje obě dvě předchozí verze. Kombinuje tedy veřejné i soukromé cloudy, které navenek působí jako jeden celek.

Komunitní (Community cloud computing)

Model komunitního cloudu se zaměřuje na skupiny lidí, organizací či jiných spolků, které dohromady svazují určitá pravidla, bezpečnostní politika, či stejný obor zájmu.

4.5.2 Dle typu distribuce

Dle úrovně distribuovaných služeb se Cloud computing dělí na tři základní kategorie. Definice jednotlivých typů čerpají ze zdrojů [17,19,21].

Software-as-a-Service (SaaS)

Ze tří uváděných typů cloud computingu je SaaS tím nejdéle používaným. Jde o poskytování softwarového produktu jako služby. Přístup ke službě je samozřejmě stejně jako u ostatních typů přes internet nebo prostřednictvím počítačové sítě obecně. Při klasickém přístupu je běžným postupem koupě softwaru v kamenné, či internetové prodejně, následně instalace a až

poté je možné jeho používání. Na rozdíl od tohoto klasického přístupu je SaaS bez instalace a zákazník si nekupuje software samotný, nýbrž platí za přístup k němu. Aktualizace a další úkony spojené s údržbou softwaru jsou zcela mimo režii zákazníka. Jako příklad můžeme uvést třeba službu Gmail od společnosti Google, která poskytuje zákazníkům rozhraní pro jednoduché vytvoření e-mailových účtů a správu jejich e-mailových schránek prostřednictvím jednoduchého webového rozhraní.

Infrastructure-as-a-Service (IaaS)

Koncept infrastruktury jako služby v podstatě není zas tak nový. Můžeme do něj zahrnout například pronájem místa v datových centrech, což bylo běžné již od počátků těchto center. Co se však změnilo, je způsob jakým jsou tyto služby poskytovány a co vše zahrnují. Kromě samotného hostingu a datového úložiště zahrnuje IaaS také služby spojené s rozdělováním zátěže (load balancing), konektivitou a zabezpečením (firewally). Uživatel má přitom částečnou či plnou kontrolu nad těmito a jinými funkcemi operačního systému dané infrastruktury. Nemůže však přímo fyzicky zasahovat do konfigurace hardwarových zdrojů.

Platform-as-a-Service (PaaS)

Tento model poskytování služeb je založen na kompletním přístupu k cílové platformě z hlediska aplikačního. Uživateli jsou vývojáři aplikací, kterým jsou poskytnuty potřebné výpočetní zdroje (dle individuální konfigurace) pro hostování svých aplikací. Zákazník nemá žádnou kontrolu nad infrastrukturou pokrývající servery, sítě, firewally, operační systémy a další, ale má plnou kontrolu nad svými aplikacemi. Může mít také částečnou kontrolu nad různými aplikačně specifickými nastaveními. Dnes se PaaS převážně zaměřují na webové služby a aplikace. Mohou však poskytovat aplikační prostředí pro hostování aplikací napsaných v různých programovacích jazycích.

4.6 Analýza dostupných řešení

Poskytovatelů veřejného cloudu již v dnešní době existuje celá řada. Uveďme alespoň některé, z těch známějších jmen. Jedná se o značky jako Microsoft, Amazon, IBM, Google a mnoho dalších. V tuto chvíli se soustředíme konkrétně na poskytovatele PaaS, což však nevylučuje, že by tito poskytovatelé nemohli nabízet i jiná řešení cloudu.

Amazon EC2 (Amazon Elastic Compute Cloud)

Jedná se o jednu ze služeb společnosti Amazon, podporující provoz aplikací, které si vyžadují širokou škálovatelnost a rychlou přizpůsobivost měnícím se požadavkům. Služba umožňuje zákazníkovi mít plnou kontrolu nad svými výpočetními zdroji a provozovat aplikace ve stabilním prostředí serverů společnosti Amazon. Tato platforma je určená především pro webové aplikace. Z těch nejznámějších můžeme jmenovat například Instagram, který právě kvůli obrovskému a neustále se měnícímu počtu aktivních uživatelů vsadil na provoz aplikace v cloudu společnosti Amazon. Kromě placených služeb nabízí Amazon novým zákazníkům balíček služeb zdarma na 1 rok. V tomto balíčku je kromě dalšího obsaženo 750 hodin provozu aplikací, 30 GB standardního diskového úložiště a 2 miliony IO operací, vše na každý měsíc provozu[13].

Amazon S3 (Amazon Simple Storage Service)

Na rozdíl od předchozího zástupce téže firmy staví Amazon Simple Storage Service na konceptu čistě datového úložiště. Uživatel nemá žádný přístup k souborovému systému a nemůže spouštět aplikace. Má k dispozici pouze pronajatý prostor datového úložiště, v rámci kterého může provádět typické operace typu zápis, čtení, apod [20]. Podobně jako u jejího předchůdce, i u této služby nabízí Amazon balíček na rok používání zdarma, který mimo jiné zahrnuje 5 GB diskového prostoru, 20.000 GET požadavků, 2.000 PUT požadavků a 15 GB přenesených dat na každý měsíc používání[14].

Microsoft Azure

Jde o veřejnou cloudovou platformu, umožňující snadné a rychlé nasazení a škálování aplikací, které mohou být vytvořeny v libovolném programovacím jazyce, nástroji a prostředí. Výhodami jsou možnost hostování aplikací v rámci globální sítě datacenter společnosti Microsoft, dále pak již zmíněná možnost škálování, a to jak pro plánovanou, tak i neplánovanou zátěž, a v neposlední řadě vysoká dostupnost služeb (SLA 99,9%)[15]. Jak sám název vypovídá, je tato cloudová platforma založena na operačním systému Microsoft Windows a využívá klasických webových technologií, jako např. SOAP, REST, ASP a aplikačních rozhraní, jako třeba Microsoft Silverlight [20]. Microsoft

[13] Zdroj: http://aws.amazon.com/ec2/pricing/.
[14] Zdroj: http://aws.amazon.com/s3/pricing/.
[15] Zdroj: http://www.microsoft.com/cs-cz/server-cloud/windows-azure.aspx.

Azure nabízí zkušební verzi v hodnotě 200 dolarů na období jednoho měsíce, umožňující plné využití všech poskytovaných služeb.

Google Cloud

Cloudová platforma společnosti Google poskytuje širokou škálu využití díky mnoha funkcím a službám, které poskytuje. Spíše než o konkrétní PaaS se jedná o množinu různých cloudových služeb. Z těchto služeb můžeme jmenovat např. Cloud SQL, Cloud Datastore, Cloud Storage, Compute Engine, či App Engine. Společnost Google láká klienty tím, že nabízí možnost používání infrastruktury, na které běží řada celosvětově úspěšných projektů a také většina aplikací firmy samotné. Google disponuje rozsáhlou globální sítí výpočetních zdrojů a datacenter, díky kterým může zajistit optimální výkon pro vysoce spolehlivé aplikace. Přitom slibuje velmi dobrou míru SLA, která např. u Google Compute Engine dosahuje asi 99,95%[16].

Právě díky postavení společnosti Google, vysoké míře dostupnosti, robustní infrastruktuře a výborné dokumentaci jednotlivých služeb, bylo pro další návrh aplikace dle zadání této práce vybráno PaaS platformy Google App Engine. Prostřednictvím této platformy bude vytvořena tzv. backend aplikace, komunikující a spolupracující s nativní mobilní aplikací v prostředí Android OS. Dále se předpokládá využití dalších služeb, jako např. Google Cloud Datastore a různých aplikačních rozhraní, které Google nabízí (Users API, Search API, apod.).

[16] Zdroj: https://cloud.google.com/.

4.7 Google App Engine

V předchozích podkapitolách bylo řečeno, že Google App Engine je cloudovou platformou typu PaaS a dále bylo vysvětleno, co je to PaaS. Z uvedených řešení byl vybrán právě Google App Engine, na který se tato kapitola blíže zaměřuje.

Google App Engine (dále GAE) se zaměřuje na webové aplikace, které jsou provozovány na infrastruktuře společnosti Google podobně jako většina z jejích veřejných webových služeb. Pozornost uživatelů se přitom zaměřuje pouze na samotný vývoj aplikací, jelikož GAE sám zabezpečuje vždy optimální výkon v závislosti na požadavcích dané aplikace a v závislosti na počtu uživatelů, kteří ji využívají. Podporovány jsou veškeré aplikace v jazycích Python, Java, PHP nebo GO s podporou databáze Cloud SQL, kompatibilní s MySQL. Tyto aplikace nasazené na GAE pak dokáží obsloužit až 7 miliard požadavků denně, což je dostačující nejen pro jednoduché, ale i pro profesionální a komplexní aplikace[17]. Ve prospěch GAE však nemluví pouze tato čísla a údaje, ale především miliony uživatelů a nasazených aplikací, které v současné době GAE má. V souvislosti s dostupnými technologiemi, kterými společnost Google disponuje, jsou uživatelům jejích služeb vždy dostupné ty nejnovější technologie a možnosti.

Pro uživatelská a aplikační data existuje hned několik možností, které se dají při práci s GAE použít. Jedná se o dříve zmíněné služby Cloud SQL, Cloud Storage a Cloud Datastore. V prvním případě se jedná o typickou SQL relační databázi, která je kompatibilní s MySQL. Druhá zmíněná služba Cloud Storage slouží pro ukládání velkých objektů a souborů do velikosti v řádech několika terabytů. Obě tyto služby jsou přes aplikační rozhraní dostupné pro implementaci v jazycích Java, PHP a Python.

Posledním zmíněným je Cloud Datastore. Jedná se o koncept datového úložiště založeného na principu NoSQL databáze, která roste dle potřeb aplikace s narůstajícím objemem dat. Detailní charakteristika NoSQL technologie by byla sama o sobě na samostatnou práci. Uveďme tedy alespoň zásadní rozdíl od typického relačního modelu databáze. Na rozdíl od klasické databáze relačního paradigmatu, kde jsou data organizována do tabulek podle klíčů a atributů, tato relativně nová forma databáze ukládá celé objekty. Tyto objekty jsou definovány svým typem a obsahují určité parametry. Jedná se tedy o jistou podobu objektově-orientované databáze. Na rozdíl od relační databáze zde není možné vykonávat některé složitější příkazy, např. typu JOIN. Google Cloud Datastore dále podporuje provádění transakcí, skládajících se z několika operací, přičemž provedení těchto transakcí je atomické. Tedy buď je transakce provedena korektně celá, nebo vůbec[18].

Pro dotazování nad Cloud Datastore byl vyvinut dotazovací jazyk GQL (Google Query Language), který vychází ze standardu SQL a je optimalizován pro použití v Cloud Datastore.

---

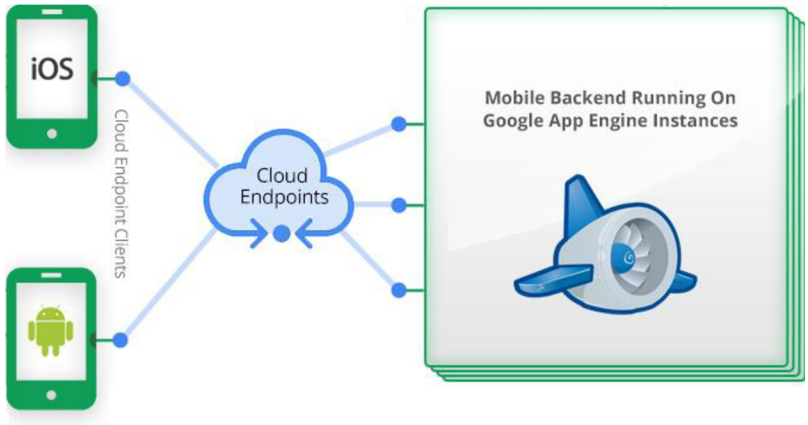
[17] Zdroj: https://developers.google.com/appengine/?hl=cs#getstarted-framework-flask.
[18] Zdroj: https://developers.google.com/appengine/docs/whatisgoogleappengine?hl=cs.

5 Návrh aplikace pro podporu projektového managementu

Následující kapitola se věnuje návrhu vlastního řešení aplikace na podporu projektového řízení. Postupně jsou uvedeny klíčové návrhové diagramy modelovacího jazyka UML 2.0, vytvořené v prostředí aplikace Visual Paradigm for UML[19]. Diagramy zachycují nejdůležitější části navrhované aplikace s ohledem na postupy používané v projektovém managementu.

5.1 Specifikace požadavků

Cílem této kapitoly je popsat návrh aplikace umožňující snadné sledování a řízení průběhu projektu, definici činností, správu zdrojů a přiřazovaní rolí. Dále je úkolem pro tento návrh využít spojení s vybraným PaaS cloudovým řešením, prostřednictvím kterého budou data synchronizována mezi uživateli mobilní aplikace. Použitím cloudu navíc bude dosaženo snadného škálování při případném rostoucím počtu aktivních uživatelů aplikace. Jak bylo zmíněno v přechozí kapitole, pro implementaci byl vybrán Google App Engine, který umožňuje prostřednictvím tzv. Cloud Endpoints jednoduše vytvářet mobilní frontend aplikace, komunikující s backend instancemi aplikací běžícími na GAE. Prostřednictvím této technologie lze snadno vytvářet aplikace jak pro Android OS, tak pro Apple iOS, jak je znázorněno na *Obrázek 10*. Součástí této práce je však pouze návrh a implementace pro zařízení platformy Android.



Obrázek 10. Abstraktní model spolupráce Android zařízení s GAE backendem[20].

[19] Oficiální web: http://www.visual-paradigm.com/.
[20] Zdroj: https://cloud.google.com/developers/articles/how-to-build-mobile-app-with-app-engine-backend-tutorial.

Hlavním požadavkem je jednoduché a přehledné intuitivní ovládání. Co se týče samotné organizace, bude aplikace rozdělena do několika kategorií, viz *Obrázek 11*. Každá z těchto kategorií je specifická svým obsahem a různou úrovní oprávnění podle funkce, kterou přihlášený uživatel v daném projektu zastává. V rámci portfolia, které je nejvyšší jednotkou v hierarchii spravovaných dat, existují tyto čtyři funkce: *manažer portfolia*, *projektový manažer*, *vedoucí etapy*, *řadový zaměstnanec*. Následuje výčet jednotlivých rolí z hlediska oprávnění a výčet k nim příslušejících oblastí spadajících pod tato oprávnění.

Manažer portfolia • Projektové portfolio • Správa projektového portfolia • Správa zaměstnanců • Správa projektů	***Projektový manažer*** • Správa projektu • Správa zdrojů
Vedoucí etapy • Správa etapy • Deník zaměstnance	***Pracovník*** • Deník zaměstnance

Tabulka 3. Rozdělení funkcí jednotlivých rolí.

Manažerem portfolia je automaticky určena osoba, která nové portfolio založí. Tato osoba je pak zodpovědná za zakládání nových projektů, přiřazování pracovníků na tyto projekty, určování projektových manažerů a celkovou správu portfolia a jeho projektů z pohledu nejvyššího managementu.

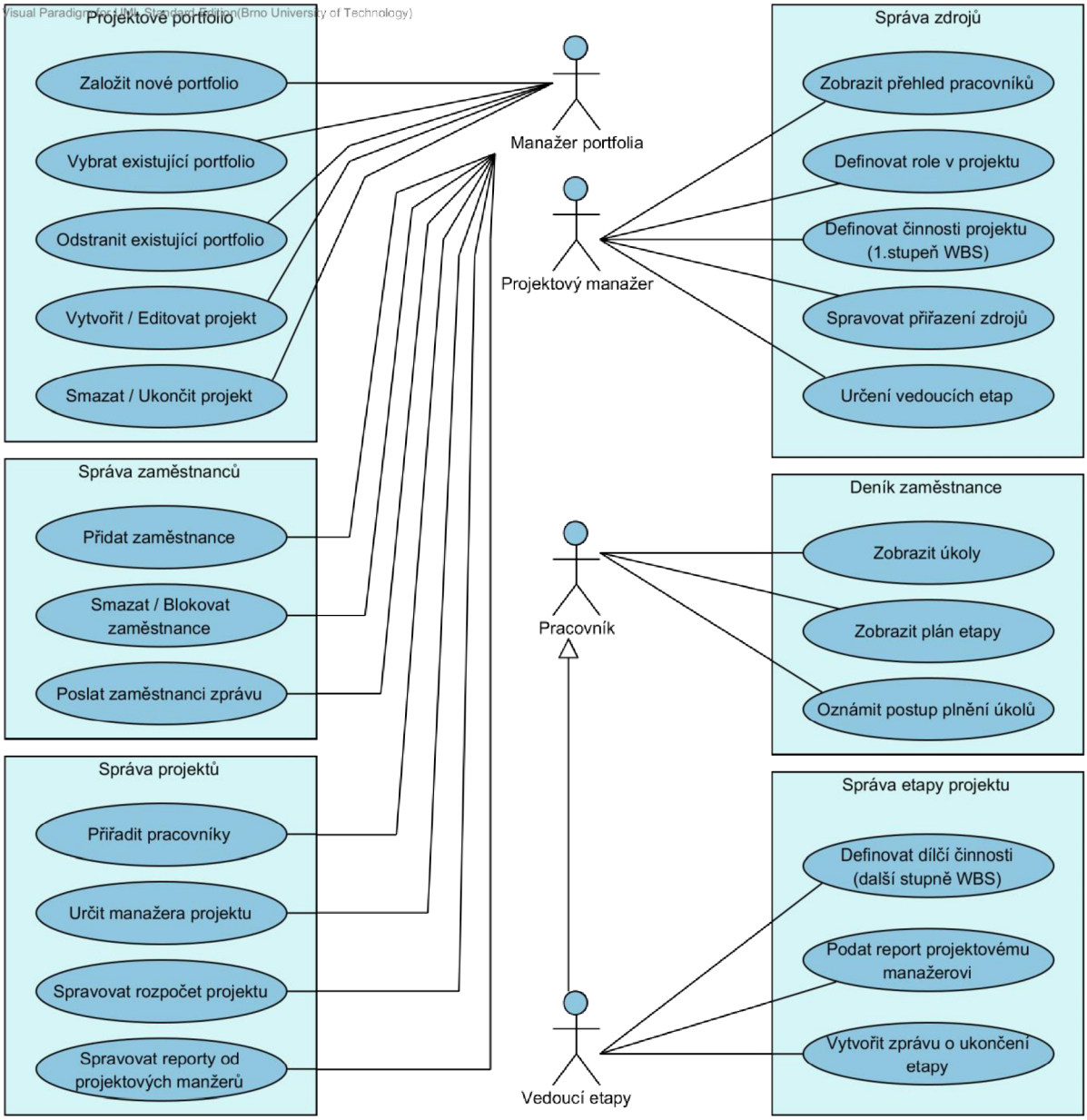
Projektový manažer je uživatel s oprávněními spravovat konkrétní projekt, ke kterému je přidělen manažerem portfolia. Jeho úkolem je definovat jednotlivé fáze projektu a příslušné činnosti časového harmonogramu. Dále disponuje dostupnými zdroji projektu, kterým přiřazuje role a zadává práci na jednotlivých činnostech. Důležitým úkolem je rovněž stanovení vedoucích etap.

Vedoucí etapy je pracovník, který je zodpovědný za případné další rozložení činností na dílčí aktivity. Zodpovídá za plnění úkolů své etapy a za její včasné dokončení. Podává zprávu projektovému manažerovi.

Na nejnižším stupni hierarchie je *řadový pracovník*, který je zodpovědný pouze za vykonání přidělené práce ve stanoveném termínu. Má k dispozici přehled časového plánu projektu a obrazovku pro správu svých činností.

5.2 Diagram případů použití

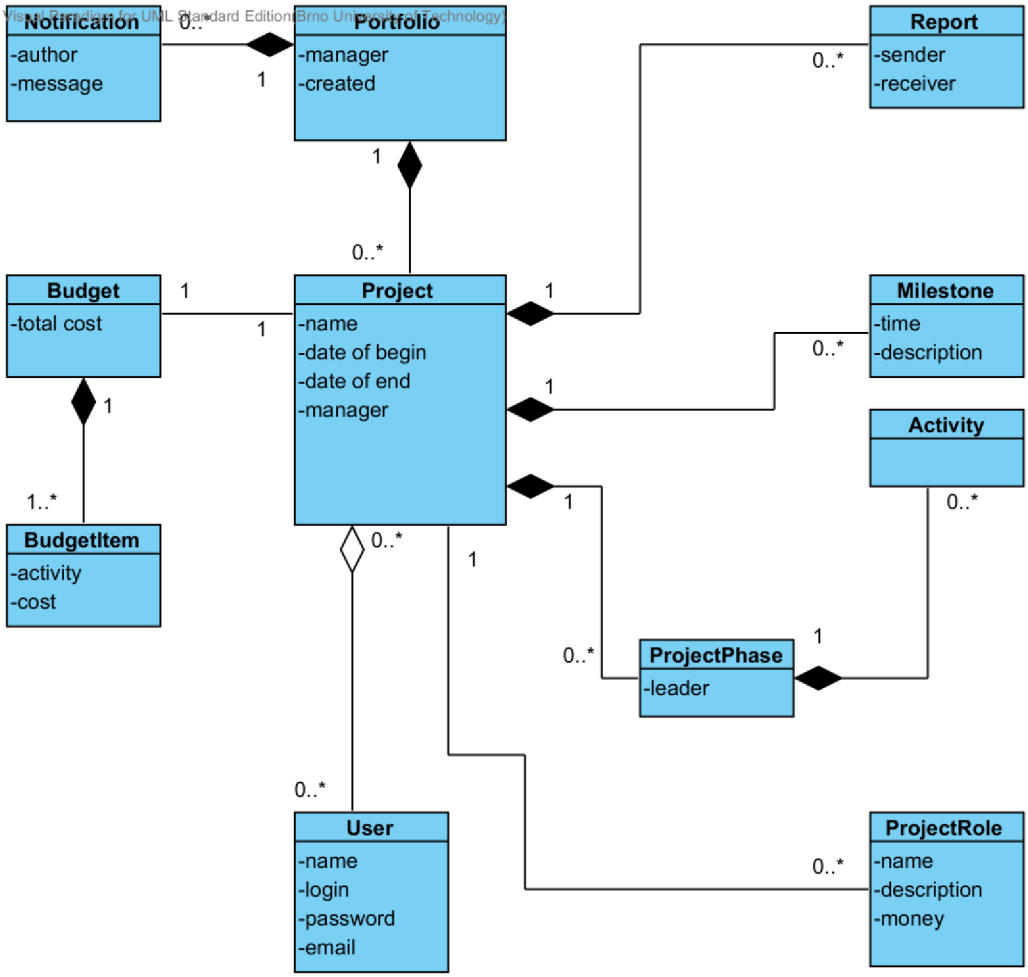
Následující diagram uvedený na *Obrázek 11* zachycuje jednotlivé případy použití, vyplývající z předchozí specifikace požadavků. Kontejnery ohraničující skupiny případů použití odpovídají rozdělení aplikace na logicky související oblasti. Dále je v diagramu použito vztahu typu generalizace-specializace, který je využit pro vyjádření vztahu mezi *Vedoucím etapy* a *Pracovník*. Tato vazba říká, že *Vedoucí etapy* má stejná práva jako řadový *Pracovník*, avšak navíc má odpovědnosti spojené s řízením průběhu celé etapy projektu.



Obrázek 11. Diagram případů použití.

5.3 Konceptuální návrh

Z definovaných požadavků a stanovených případů použití lze určit klíčové entity aplikace. Tyto entity a vztahy mezi nimi jsou vyjádřeny diagramem na *Obrázek 12*. Mezi tři hlavní prvky, které v projektu přímo vystupují, patří *Project*, *Portfolio* a *User*. Vztah těchto a dalších entit je v diagramu vyjádřen pomocí tří vztahů – asociace, agregace, kompozice. Dále jsou na vstupních a výstupních koncích hran uvedeny násobnosti vyjadřující počet instancí entit, které do tohoto vztahu mohou vstupovat.

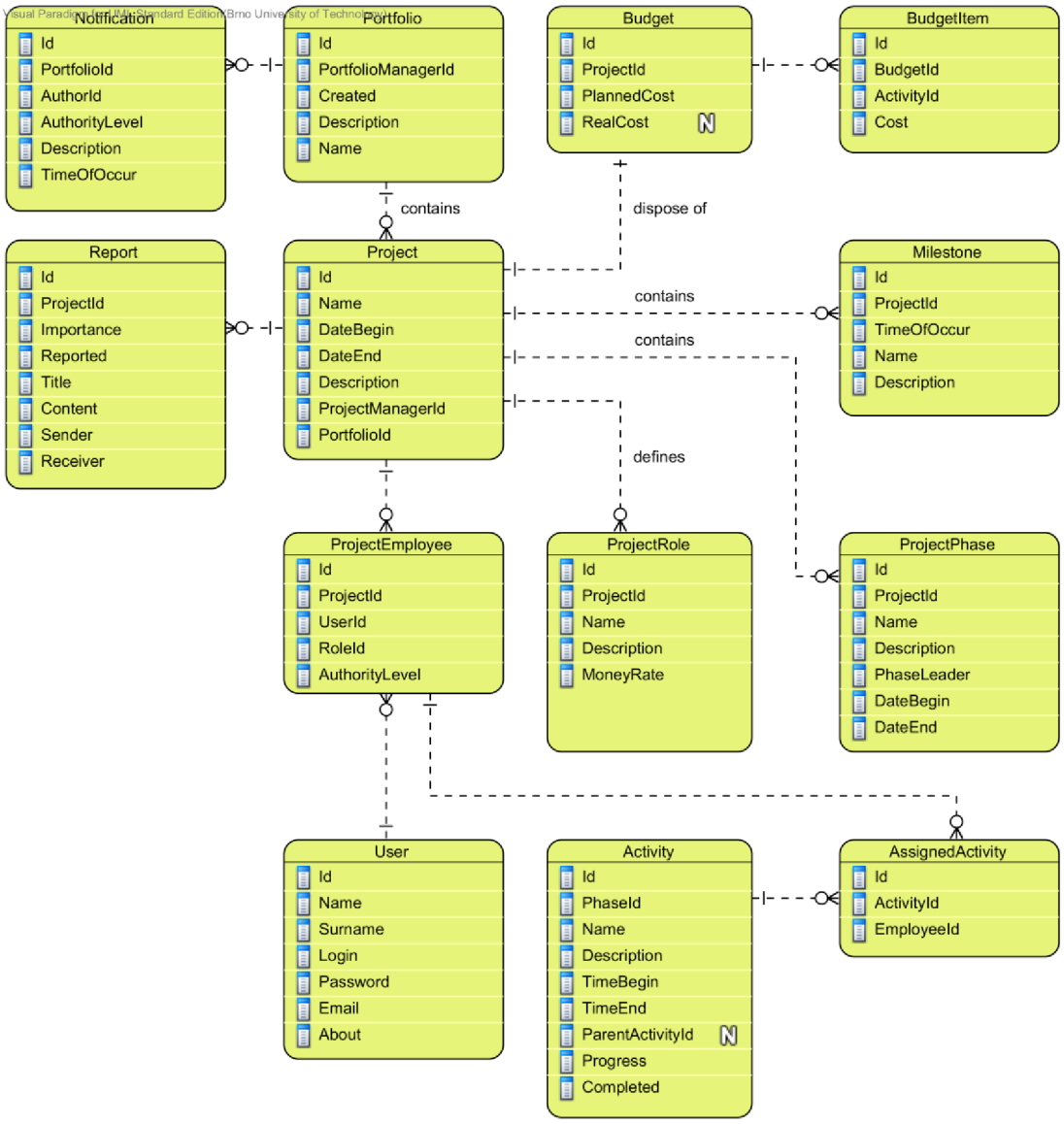


Obrázek 12. Konceptuální návrh aplikace.

Z tohoto diagramu pak lze vycházet při návrhu databáze, kterým se zabývá následující podkapitola.

5.4 Návrh databázové struktury

Na základě konceptuálního modelu z předchozí podkapitoly byl vytvořen návrh struktury databáze. Přestože se nejedná o klasickou podobu relační databáze, nýbrž o databázi typu NoSQL, je možné využít tohoto diagramu pro zachycení vztahů mezi jednotlivými objekty, podobně jako u konceptuálního modelu. Oproti předchozímu modelu je však nutné v případech vztahů M:N použít třetí entitu, podobně jako při vytváření vazebních tabulek v relační databázi. Příkladem této modifikace je např. v níže uvedeném diagramu entita *ProjectEmployee* nebo *AssignedActivity*. U všech zobrazených entit jsou uvedeny nezbytné atributy pro základní funkčnost aplikace. Není vyloučeno, že v průběhu implementace a po prvních uživatelských zkušenostech budou doplněny některé další atributy související s rozšiřováním funkcionality aplikace.

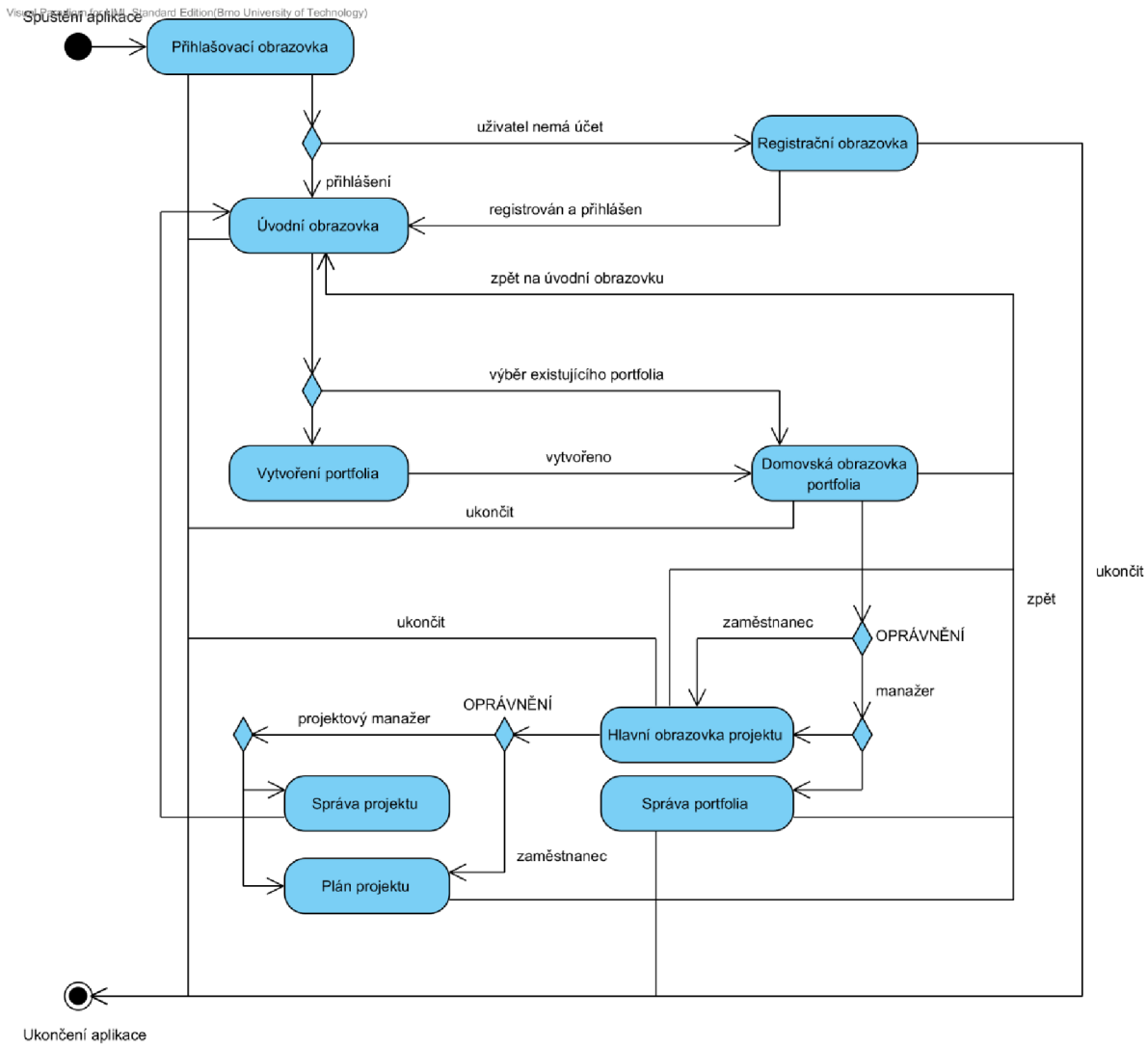


Obrázek 13. Návrh struktury databáze aplikace.

5.5 Diagram návaznosti obrazovek

Pouze pro základní představu o navigaci skrze jednotlivé obrazovky aplikace slouží následující diagram aktivity. Jsou na něm zachyceny hlavní obrazovky, které by měly být v zájmu uživatele. Obrazovky jsou v tomto diagramu aktivity reprezentovány modrými ovály. Na hranách jsou pak uvedeny uživatelské interakce, které lze z dané obrazovky provést (opět se jedná pouze o ty nejdůležitější). Dále jsou uvedeny tzv. rozhodovací uzly (modrý kosočtverec), které slouží pro vyjádření větvení na základě možné volby z více možností, či na základě různého oprávnění uživatele. Z těchto hlavních obrazovek bude dále možná navigace na další podsekce a skryté obrazovky prostřednictvím menu aplikace a menu aktivní obrazovky.

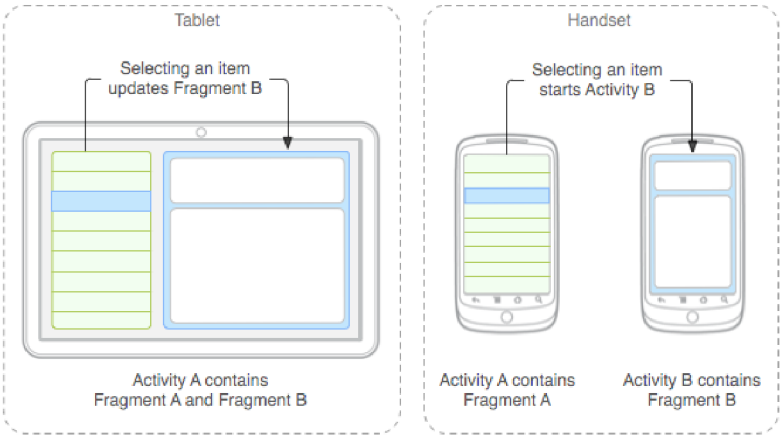
Náčrt rozložení obrazovek aplikace a grafický návrh přihlašovací obrazovky včetně návrhu loga aplikace je součástí další podkapitoly.



Obrázek 14. Diagram aktivity vyjadřující návaznost obrazovek.

5.6 Grafický návrh

Aplikace je primárně navržena pro tablety s širokou úhlopříčkou. Použitím tzv. fragmentů by však měla být zachována kompatibilita i pro jiná zařízení (většina telefonů), která mají mnohem menší rozměry displeje a vertikálně orientovaný displej. Tento princip je znázorněn na *Obrázek 15*. Levý sloupec bude obsahovat kontextové menu odkazující na aktivity, které se budou zobrazovat v pravém sloupci (u telefonů viz obrázek).



Obrázek 15. Rozvržení obrazovky aplikace na fragmenty[21].

Celkový vzhled aplikace bude laděn do kombinace barevných odstínů červené, černé, šedé a bílé barvy. Jednotlivé části aplikace by tak měly odpovídat barevnému konceptu navrženého loga, které je uvedeno na *Obrázek 16*, který zároveň představuje přihlašovací obrazovku aplikace. Prvotní návrh loga byl proveden na papíře, poté došlo k jeho digitalizaci pomocí programu Adobe Photoshop.

Obrázek 16. Grafický návrh přihlašovací obrazovky s logem aplikace (Aybee).

[21] Zdroj: http://developer.android.com/guide/components/fragments.html.

6 Implementace

Kapitola implementace se věnuje realizačním detailům praktické části diplomové práce. Uvádí hlavní a nejdůležitější části aplikace od abstraktního pohledu na aplikaci jako celek, až po konkrétní třídy použitého jazyka Java.

6.1 Popis aplikace

Jak již bylo zmíněno, mobilní aplikace pro platformu Android byla implementována dle předchozího návrhu s několika málo odchylkami od původního plánu. Tyto změny budou dále v textu uvedeny. Vzhledem k současnému vývoji mobilních zařízení, který jde rychle kupředu, byla implementace zaměřena na vývoj aplikace kompatibilní s verzemi Android 4.0 a vyšší. Jelikož většina tabletů dnes používá právě verze 4.0, 4.1, 4.2 tohoto operačního systému, nebylo nutné zaměřovat se na zpětnou kompatibilitu pro starší verze Android. Aplikace by sice měla fungovat i na starších verzích, nicméně tato skutečnost nebyla testována.

Pro vytvoření aplikace bylo použito vývojového prostředí Eclipse s pluginy ADT Bundle pro vývoj na platformě Android a „Google plugin“ pro vývoj webových aplikací na platformě Google (GWT, Google App Engine, a další). V prostředí Eclipse pak lze snadno založit projekt přímo určený pro vytváření propojených aplikací (Android + Google App Engine). Postup vytvoření takového projektu je zdokumentován na oficiálních stránkách vývojářů Google[22]. Co se týče projektů pro aplikace na Android, byla jejich struktura již popsána ve druhé kapitole věnující se této platformě. Struktura serverové části, tedy aplikace běžící na Google App Engine, bude vývojářům webových aplikací v jazyce Java jistě známá.

6.2 Serverová část

V projektu se opět vyskytuje adresář `src`, obsahující zdrojové soubory s třídami jazyka Java. V našem případě se jedná o třídy reprezentující datové entity s danými anotacemi dle zvoleného API (JPA, JDO, low-level API). Při implementaci této práce bylo zvoleno JDO API. Jedná se o aplikační rozhraní vycházející z jazyka Java, které slouží pro uchování perzistentních dat ve formě tříd tohoto jazyka. Přitom není zapotřebí, aby tyto třídy implementovaly nějaké specifické rozhraní nebo rozšiřovali jiné třídy. Jde tedy o běžné třídy označené anotacemi, které slouží pro popis perzistentních dat a jejich konfiguraci. Tyto anotace mohou být aplikovány na celé třídy nebo na jednotlivá datová pole. Seznam použitelných anotací včetně jejich významu je uveden například na stránkách platformy DataNucleus[23], kterou právě Google App Engine využívá pro manipulaci s daty. Kromě těchto

[22] Web: http://developers.google.com/eclipse/docs/endpoints-androidconnected-gae.
[23] Web: http://www.datanucleus.org/products/datanucleus/jdo/annotations.html.

souborů reprezentujících datové entity jsou dále v adresáři `src` umístěny soubory, obsahující metody pro manipulaci s danými entitami. Google plugin v prostředí Eclipse umožňuje tyto soubory generovat automaticky kliknutím pravým tlačítkem myši na vybranou třídu a volbou možnosti *Google > Generate Cloud Endpoint Class*. Jinou možností je vytvoření těchto tříd manuálně, avšak v tomto případě je nutné velmi dobře rozumět tomu jakým způsobem takovéto třídy tvořit a jak dané metody principiálně fungují. Metody obsažené v těchto třídách jsou pak použity pro generování tzv. *endpoints*, tedy do češtiny volně přeloženo koncových bodů, které slouží pro komunikaci mezi klientem na straně Android, či iOS aplikace a serverem (backendem) na straně Google App Engine. Kromě datových entit a koncových bodů se samozřejmě v adresáři `src` mohou nacházet další zdrojové soubory pro implementaci funkcí backendové části aplikace.

6.2.1 Datové entity v JDO

V předchozím textu byla zmíněna zkratka JDO, pod kterou se skrývá název *Java Data Objects*. Pomocí JDO lze prostřednictvím anotací definovat mapování tříd v jazyce Java na perzistentní data, která jsou po uložení reprezentována ve formě entit v Google Cloud Datastore. Navíc lze pomocí anotací JDO data nejen ukládat, ale i zpětně načítat z datového úložiště a ze získaných dat znovu vytvářet objekty s příslušnými vlastnostmi. U těchto datových tříd rozlišujeme mezi anotacemi, kterými se označují celé třídy, a anotacemi, které slouží pro popis jednotlivých vlastností dané třídy. Následující text čerpá z oficiální dokumentace dostupné z [23].

Pro označení tříd, které mají být ukládány a zpětně načítány z Google Cloud Datastore, je nutné uvést anotaci `@PersistenceCapable` týkající se celé třídy, tak jak je uvedeno v následující ukázce zdrojového kódu.

```
import javax.jdo.annotations.PersistenceCapable;

@PersistenceCapable
public class Employee {
    // ...
}
```

Takto anotovaná třída je tedy připravená pro uložení do datového úložiště, nicméně ještě je zapotřebí podobným způsobem označit všechna pole dané třídy, která mají být uložena jako vlastnosti dané entity. Pro tyto účely se využívá anotace `@Persistent`, která se uvádí před deklarací daného pole, viz následující příklad.

```
import javax.jdo.annotations.Persistent;

// ...
    @Persistent
    private String employeeName;
```

Pro deklaraci pole, které nemá být ukládáno do datového úložiště, je nutné ho označit anotací `@NotPersistent`. Je vhodné vždy uvádět anotace u všech polí, ať už se jedná o ta, která mají být uložena, či ta která uložena být nemají. Pro pole určitých datových typů je totiž použito výchozího nastavení, kdy jsou automaticky považována za pole určená k perzistenci. Tato pole mohou být různých typů, přičemž podporovány jsou následující:

- Datové typy podporované Google Cloud Datastore (Integer, String, Date, Boolean, a další)
- Datový typ kolekce (např. `List<…>`) nebo pole hodnot podporovaných datových typů
- Instance nebo kolekce instancí tříd označených jako `@PersistenceCapable`
- Instance nebo kolekce instancí tříd `Serializable`
- Vestavěná třída, uložená jako vlastnost dané entity (např. kontaktní informace)

V implementaci naší aplikace bylo použito prvních tří variant. Tedy aplikace pracuje se základními datovými typy jako `String`, `Integer`, `Date`, `boolean` a dále především s kolekcemi definovaných perzistentních tříd. Jako kolekce je přitom použit datový typ `List<>`. Tímto způsobem lze mezi entitami vytvářet relace typu 1:N, případně M:N, detaily budou popsány později.

Dále musí každá třída reprezentující entitu v datovém úložišti mít jedno pole, které představuje primární klíč této entity. V Google Datastore se doporučuje používat datový typ `Key`, který je vhodný pro automatické generování unikátních klíčů v době ukládání entity. Toto pole je rovněž opatřeno anotací `@Persistent` s parametrem `valueStrategy = IdGeneratorStrategy.IDENTITY`, který říká, že klíč má být generován až při vkládání do datového úložiště. Že se jedná o primární klíč je navíc uvedeno anotací `@PrimaryKey`, viz následující příklad.

```
import com.google.appengine.api.datastore.Key;

import javax.jdo.annotations.IdGeneratorStrategy;
import javax.jdo.annotations.PrimaryKey;

// ...
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key key;
```

6.2.2 Relace mezi entitami

Použité datové úložiště Google Cloud Datastore je typem databáze NoSQL, tedy nejedná se o klasickou relační databázi. Nicméně jak z návrhu databázové struktury z předchozích kapitol vyplývá, je nutné mezi vybranými entitami určité relace modelovat. Nejpoužívanějším typem relace v aplikaci je model 1:N. Již bylo zmíněno, že pro tyto účely jsou použity kolekce, konkrétně datový typ `List<>`. Ve vazbě na databázový návrh z páté kapitoly této práce můžeme uvést příklad relace 1:N na vztahu mezi entitami `Portfolio` a `Project`.

Požadovaný vztah je zřejmý. Chceme uchovávat informace o různých portfoliích, přičemž každé z těchto portfolií bude obsahovat vlastní množinu projektů. Jelikož projekty vznikají právě v kontextu vybraného portfolia, je potřeba v relaci řešit i možnost, kdy dojde ke smazání daného portfolia. V takovém případě pro nás nemá význam dále uchovávat příslušné projekty a můžeme je tedy rovněž odstranit. Vztah tohoto typu se v Google Cloud Datastore nazývá tzv. *owned relationship* a lze jej modelovat využitím anotace `@Element` s parametrem `dependent = "true"`. Dále je žádoucí, aby tato kolekce projektů byla nějakým způsobem seřazena. V ukázce níže lze vidět použití dalšího typu anotace a sice `@Order` s příslušnými parametry. Parametr `key` anotace `@Extension` říká, že požadujeme řazení v kolekci typu `List<>` a parametr `value` udává, podle jakého pole má být kolekce seřazena. `ASC` pak značí řazení vzestupně. Předpokládejme tedy existenci třídy `Project` anotované pomocí `@PersistenceCapable`. Vztah 1:N mezi entitou `Portfolio` a `Project` s řazením a s označením závislosti by pak mohl vypadat následovně:

```
@PersistenceCapable
public class Portfolio {
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key key;

    @Persistent
    @Element(dependent = "true")
    @Order(extensions = @Extension(vendorName = "datanucleus",
      key = "list-ordering", value = "name ASC"))
    private List<Project> projects;

    // ...
}
```

Podobný vztah jako u tohoto případu je v aplikaci použit mezi následujícími dvojicemi entit (první uvedený značí rodičovský objekt, druhý značí potomka): `Project -> ProjectEmployee`, `Project -> ProjectRole`, `Project -> ProjectPhase`, `Project -> Report`, `ProjectPhase -> Activity`, `Portfolio -> PortfolioEmployee`.

6.2.3 Třídy Google Cloud Endpoints

Základem Google Cloud Endpoints je globálně rozšířené a ve webových aplikacích hojně využívané REST[24] API. Jedná se o jednotný a snadný přístup ke zdrojům v distribuovaném prostředí, přičemž zdroji zde mohou být např. data nebo různé stavy aplikací. Na rozdíl od RPC a SOAP je tedy REST orientován spíše datově, nežli procedurálně. REST rozhraní je zde použito pro implementaci tzv. CRUD[25] operací [20] s využitím základních HTTP metod. Na těchto primitivních metodách jsou pak postaveny metody Google Cloud Endpoints, které v našem případě slouží pro manipulaci s daty v Google Cloud Datastore.

Jako ve většině případů, kdy je zapotřebí zpracovávat určitá data z databáze, i zde existují určité základní metody, které realizují výše zmíněné CRUD operace. Pro vytváření názvů těchto metod platí nepsané pravidlo, které označuje metody pro získávání dat z databáze klíčovým slovem ***get*** (získání konkrétního záznamu), případně ***list*** (získání kolekce záznamů). Tyto metody využívají http metody GET. Dále se pro přidávání záznamů do databáze využívá metod začínajících klíčovým slovem ***add***, které jsou postaveny na http metodě POST. Dalším typem metod jsou ty, které slouží k aktualizaci záznamů. Většinou se značí klíčovým slovem ***update*** a využívají http metody PUT. Posledním typem operace je smazání záznamu. Příslušné metody začínající klíčovým slovem ***delete*** případně ***remove***. Tyto metody pak užívají http metody DELETE. Za klíčovým slovem dané operace bývá následně uveden název dané entity.

Uvedený způsob je pouze vhodným doporučením pro tvorbu metod. Při dodržení tohoto postupu je však v Eclipse možné jednoduše generovat klientské třídy Google Cloud Endpoints bez toho aniž by bylo nutné uvádět použité http metody. Google plugin se o mapování na patřičné metody postará sám, právě prostřednictvím výše uvedených klíčových slov. V případě, že třídy koncových bodů tvoříme manuálně, je nutné u každé z metod, která má být viditelná na straně klienta uvést anotaci označující jméno dané metody a v případě použití nestandardních názvů metod i anotaci pro rozlišení použité http metody. V následující ukázce kódu lze navíc vidět i anotaci `@Named` u parametru uvedené metody `getEmployee`, která slouží pro pojmenování parametru, jehož jméno bude použito na straně klienta.

```
import javax.jdo.annotations.PersistenceCapable;

@ApiMethod(name = "getEmployee", httpMethod = "GET")
public Employee getEmployee(@Named("employeeId") Long id) {…}
```

[24] REST = Representational State Transfer.
[25] CRUD (C = vytvoření, R = čtení, U = update, D = odstranění).

Ať už se jedná o jakoukoliv z metod typu `get`, `list`, `add`, `update`, `remove`, všechny používají tzv. *persistence manager*. Jedná se o instanci třídy, která slouží pro vykonávání operací nad datovým úložištěm. V celém projektu, tedy ve všech třídách „endpoint“, se pracuje vždy pouze s jedinou instancí této třídy, která je vytvořena pomocí třídy *JDO Helper* a jejíž instance je uchována v objektu třídy `PMF.java`.

```
import javax.jdo.JDOHelper;
import javax.jdo.PersistenceManagerFactory;

// ...
private static final PersistenceManagerFactory pmfInstance = JDOHelper
            .getPersistenceManagerFactory("transactions-optional");
```

Použití v rámci metod je pak jednoduché. V každé z uvedených metod je třeba získat instanci `PersistanceManager`, nad kterou jsou volány metody typu `makePersistent()`, `deletePersistent()`, `getObjectById()`, `close()`, `detachCopy()`, apod.

```
@ApiMethod(name = "getUser", path = "getUser")
public User getUser(@Named("id") Long id) {
      PersistenceManager mgr = getPersistenceManager();
      User user = null;
      try {
            user = mgr.getObjectById(User.class, id);
      } finally {
            mgr.close();
      }
      return user;
}
```

6.2.4 Nastavení Google App Engine

Pro vývoj aplikací na Google App Engine (GAE) je nutné mít založený účet Google. Vytvoření nového účtu je otázkou maximálně několika málo minut. S existujícím účtem můžeme po přihlášení ke službám Google přejít na domovskou stránku platformy Google Cloud[26]. Pro začátečníky nebo vývojáře, kteří si chtějí GAE pouze vyzkoušet a prozkoumat jeho možnosti, je na této stránce uveden odkaz pro vyzkoušení, který spustí interaktivního průvodce založením a vytvořením aplikace. Pro tyto účely jsou zde k dispozici dvě různé demo aplikace, na kterých si lze osvojit základní principy tvorby aplikací v prostředí Google App Engine. Na stejné stránce se nachází i odkaz pro přechod do vývojářské konzole. Po kliknutí na tento odkaz se zobrazí úvodní stránka konzole, která obsahuje přehled všech dostupných aplikací. Na této stránce lze provádět nastavení účtování, nastavení jazyka, měny apod.

[26] Web: https://cloud.google.com/.

Pro vytvoření nového projektu jsou vyžadovány údaje název a ID projektu. Název volíme libovolně podle uvážení. Pro nastavení a fungování aplikace není název projektu důležitý. Co je však podstatně důležitější je ID projektu, které je ve výchozím nastavení generováno automaticky, nicméně je možné zvolit vlastní, pokud je toto ID volné. Tato hodnota se nastavuje pouze jednou při vytváření projektu a dále je neměnná. Slouží pro jednoznačnou identifikaci instance dané aplikace v rámci Google App Engine.

Vytvořením nového, případně výběrem existujícího projektu se dostáváme na hlavní stránku aplikace. Zde jsou pro nás nejdůležitější dva údaje uvedené v záhlaví stránky. Jedná se o již zmíněné *Project ID* a *Project Number*. Oba tyto údaje budeme později potřebovat při konstrukci klientské aplikace komunikující s aplikací v Google App Engine. Druhou položkou v kontextovém menu v levé části obrazovky je položka *APIs & auth*. V této kategorii jsou pro nás důležité především první dvě nabídky. První z nich je sekce *API*, která obsahuje výpis všech aktivovaných/deaktivovaných API, které jsou, resp. nejsou pro danou instanci aplikace spuštěny. K tomuto nastavení se vrátíme později při konfiguraci služby Google Cloud Messaging (GCM). Druhou položkou je pak podsekce *Credentials*, která slouží k vytváření unikátních klíčů. Toto nastavení budeme rovněž používat v implementaci GCM. Co nás dále zajímá u projektu našeho typu, jsou položky *App Engine*, obsahující důležité informace o aktivitách běžící aplikace. Najdeme zde logovací záznamy, hlášení o chybách, grafy využití zdrojů, nastavení aplikace a další. Poslední, avšak neméně důležitou položkou je *Cloud Datastore*, prostřednictvím které lze spravovat datové úložiště využívané naší aplikací. Lze zde manuálně vytvářet a spravovat datové entity, provádět dotazy, spravovat indexy či sledovat využití datového úložiště pomocí přehledných grafů.

6.2.5 Google Cloud Datastore

Následující text se věnuje implementaci databázové struktury aplikace s využitím Google Cloud Datastore a čerpá informace z [24]. Rovněž uvádí vybrané zajímavé úseky zdrojového kódu aplikace.

Google Cloud Datastore (dále jen „datastore“) je datové úložiště založené na principu databáze NoSQL. Jedná se o robustní nástroj pro uchování dat, přičemž je zajištěna vysoká stabilita a spolehlivost díky distribuovanému způsobu ukládání dat na různých datových serverech společnosti Google. Data jsou uložena ve formě entit, které mohou být zpracovávány pomocí dostupných rozhraní (JDO, JPA, low-level API), či existujících frameworků jako je např. Objectify nebo Slim3.

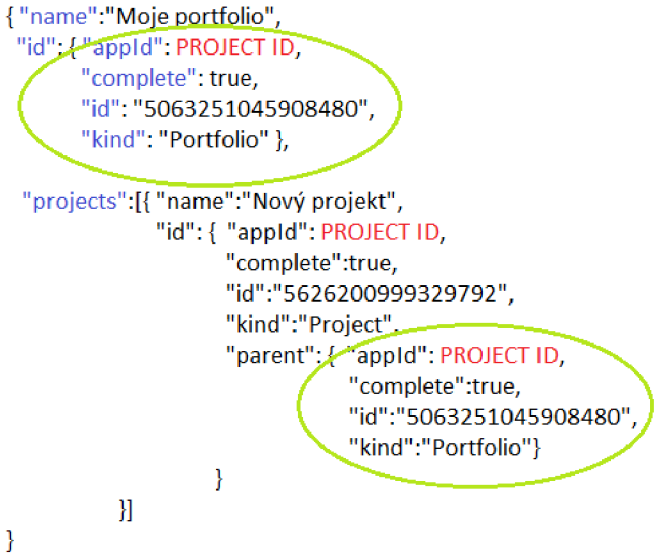
6.2.5.1 Entity, klíče, ancestor path

Každá entita sestává z klíče, který ji jednoznačně identifikuje v celém datastore. Struktura klíče je následující:

- Namespace – může být použit pro odlišení dat jedné instance aplikace běžící např. ve více firmách nebo u různých zákazníků, u kterých potřebuje data zvlášť oddělit.

- Kind – druh entity, který ji rozlišuje od ostatních druhů v datastore (analogie k názvu tabulky v relační databázi).
- Identifier – identifikátor konkrétní entity; může být typu řetězec nebo číselný.
- Ancestor path – volitelná položka klíče, která určuje pozici entity v hierarchii datastore entit (používá se pro vyjádření vztahu rodič-potomek).

Aplikace pro správu projektů, která je předmětem této práce, využívá složení klíče z atributů *kind*, *identifier* a v mnoha případech i *ancestor path*. Vzhledem k povaze aplikace, která by měla fungovat v jedné aktivní instanci, není použito atributu *namespace* pro rozdělení datastore do různých oddílů. Jména entit jsou v případě použití JDO API odvozena od názvů anotovaných tříd. Jednotlivá pole těchto tříd pak tvoří vlastnosti odpovídající entity v datastore. Při použití kolekce objektů tříd (anotovaných `@PersistenceCapable`) uvedených v poli jiné třídy jsou po uložení takovéto entity vytvořeny i entity dceřiné odpovídající položkám dané kolekce. V tomto případě dané entity obsahují kromě svého jednoznačného identifikátoru i zmíněnou *ancestor path*, ve které je nadřazená entita zaznamenána jako rodičovská s uvedením jejího jednoznačného klíče. Způsob tvoření hierarchie je uveden na následujícím obrázku, který znázorňuje formou JSON[27] objektu zjednodušenou strukturu entity `Portfolio` obsahující 1 až N různých entit typu `Project`. V tomto případě je pro přehlednost uveden pouze jeden projekt a obě entity jsou značně zjednodušeny pro lepší přehlednost. Na místě zvýrazněné hodnoty `PROJECT ID` se objevuje konkrétní ID dané aplikace, které jsme zvolili při zakládání projektu ve vývojářské konzoli a které je třeba nastavit i v prostředí Eclipse, aby při publikování aplikace na produkční server mohlo dojít ke správnému spárování. Toto nastavení, stejně jako další týkající se GAE lze provádět v menu projektu *Properties* > *Google* > *App Engine*.



Obrázek 17. Hierarchická struktura s využitím ancestor path.

[27] JSON = JavaScript Object Notation (způsob zápisu objektových dat v jazyce JavaScript formou řetězce).

Pro práci s entitami obsahujícími *ancestor path* je potřeba přistupovat k záznamům na nejnižší úrovni přes jednotlivé rodičovské identifikátory na úrovních vyšších. Například při vkládání nového projektu do portfolia je třeba nejprve získat objekt dané entity `Portfolio`. Z tohoto objektu následně získáme klíč, kterým je entita jednoznačně identifikována. Tento klíč je poté použit při vytváření nového záznamu typu `Project`, kde je uveden jako hodnota atributu `parent` v identifikátoru nového projektu (viz *Obrázek 17*). Ve vyvíjené aplikaci je tento princip použit velmi často a to v různých stupních hierarchie. Nejdelší cestou napříč hierarchií celé aplikace je následující dekompozice: `Portfolio -> Project -> ProjectPhase -> Activity -> Worker`. Mezi každými dvěma přímo sousedícími entitami pak platí vztah 1:N. Tedy portfolio může obsahovat několik projektů, projekt je členěn do několika fází, atd. Následující úryvek kódu pochází ze souboru `ProjectEndpoint.java`, tedy z třídy Google Cloud Endpoints, která se stará o manipulaci s entitami typu `Project`. Konkrétně jde o část kódu metody `getProject()`. Jak je vidět je uplatněn postup popsaný výše, tedy nejdříve je získán objekt nadřazené třídy `Portfolio`. Klíč tohoto objektu je pak použit pro získání objektu třídy `Project`.

```
// ...
Key childKey;

Portfolio pf = mgr.getObjectById(Portfolio.class, portfolioId);
childKey = pf.getId().getChild(Project.class.getSimpleName(), projectId);
Project project = mgr.getObjectById(Project.class, childKey);
```

Uvedené parametry `portfolioId` a `projectId` jsou jednoznačné číselné identifikátory příslušných entit. Za objektem `mgr` se neskrývá nic jiného než dříve zmiňovaná instance třídy `PersistenceManager`. Na uvedeném příkladu je vidět, že metoda `getObjectById()` je použita různými způsoby. Prvním způsobem je volání metody s číselným parametrem ID (v projektu je užito datového typu `long`). Tento způsob použití metody je vhodný pro entity, které nemají v hierarchii datového úložiště žádného předka. V našem případě se jedná právě o entitu `Portfolio`. Další takovou je entita `User`, v níž jsou uchovány informace o přihlášených uživatelích. Druhý způsob použití je pak aplikován právě na entity, jež jsou potomky jiných nadřazených entit.

Při vkládání nového záznamu jako potomka existující entity je postup podobný. Nejprve je získán nadřazený objekt stejným způsobem jako v předchozí ukázce kódu. Poté je získanému objektu nastavena daná hodnota a následně je nad instancí třídy `PersistenceManager` volána metoda `makePersistent()` s parametrem ukládané entity.

6.2.5.2 Transakce

Google Datastore kromě jiných možností umožňuje i použití transakcí. Jedná se o soubor operací, které jsou atomické. To znamená, že jsou vždy provedeny buď všechny operace v dané transakci, anebo žádná z operací. Operace obsažené v dané transakci se navíc mohou týkat jedné, ale i více entit různých typů. Z tohoto důvodu je vhodné používat transakce u metod, které nějakým způsobem manipulují s daty různých typů entit. Maximální doba trvání transakce je 60 vteřin. Pokud v rámci tohoto limitu není transakce řádně ukončena, dojde k jejímu vynucenému ukončení a veškeré operace, které byly realizovány, jsou navráceny zpět. Dalšími důvody pro nekorektní ukončení transakce může být interní chyba datového úložiště, či více současných přístupů ke stejné entitě. Všechny tyto události jsou ošetřeny odpovídajícími výjimkami `DatastoreTimeoutException`, `DatastoreFailureException` a `ConcurrentModificationException` [25].

V aplikaci pro správu projektů je principu transakcí využito u většiny metod tříd Google Cloud Endpoints. Použití transakcí je žádoucí především u operací pro odstranění záznamu z datového úložiště, kdy je v případě použití vztahu *owned one-to-many* zapotřebí, aby byly odstraněny všechny závislé entity a nedocházelo tak k nežádoucím nekonzistencím datového modelu. Jiným případem může být vkládání nového záznamu. I v této situaci je třeba ošetřit správné vytvoření všech příslušných entit a vztahů mezi nimi. To se opět týká především těch entit, mezi kterými jsou modelovány určité relace. Pro ilustraci použití transakce ve zdrojovém kódu datastore metody je uveden následující příklad.

```
// ...
Transaction txn = datastore.beginTransaction();
try {
    // some datastore operations
    txn.commit();
} finally {
    if (txn.isActive()) {
        txn.rollback();
    }
}
```

6.2.5.3 Indexy

Ve výchozím nastavení Google Datastore automaticky definuje indexy pro každý atribut všech entit. Tyto automaticky generované indexy jsou dostačující pro většinu jednoduchých dotazů (např. jednoduchá selekce bez parametrů, apod.). Pro složitější dotazy je zapotřebí indexy definovat manuálně v konfiguračním souboru `datastore-indexes.html`. Pokud dojde k situaci, kdy aplikace nemůže realizovat daný dotaz s využitím automaticky předdefinovaných, či manuálně definovaných indexů, skončí tato operace výjimkou `DatastoreNeedIndexException` [26]. Pro vyvíjenou aplikaci je nutné specifikovat několik indexů manuálně pomocí konfiguračního souboru. Jde o indexy týkající se entit, které obsahují jako parametry kolekce jiných entit, nad

kterými je požadováno vzestupné či sestupné uspořádání dle určitého parametru. Níže uvedený příklad uvádí definici indexu pro entitu typu `Project`. Tento index je využit při dotazech na získání kolekce projektů vybraného portfolia, přičemž je uplatněno vzestupné třídění (`direction="asc"`) dle parametru `name` (jméno projektu). Parametr `ancestor` tohoto indexu má hodnotu `true`, jelikož se jedná o podřazenou entitu objektu `Portfolio`.

```
<datastore-index kind="Project" ancestor="true" source="auto">
    <property name="name" direction="asc"/>
</datastore-index>
```

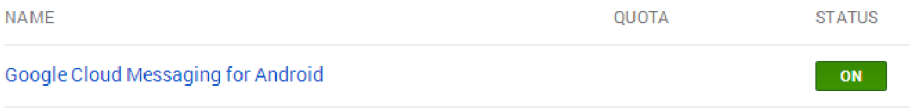
6.2.6 Google Cloud Messaging

Google Cloud Messaging (GCM) je služba, umožňující komunikaci mezi serverem běžícím na Google App Engine a mobilními zařízeními platformy Android. Tato komunikace probíhá prostřednictvím zpráv, které mohou být v rámci spojení klient-server posílány v obou směrech. Pro tuto službu nejsou stanoveny žádné limity a její používání je zcela zdarma, bez ohledu na velikost aplikace a počet požadavků [27]. Na komunikaci skrz GCM se podílí tři aktéři. Jsou to klientská zařízení s operačním systémem Android, dále GCM server, který zprostředkovává zasílané zprávy a posledním je aplikační server třetí strany, který generuje požadavky na zasílání GCM zpráv. Pojem třetí strany je v tomto případě poněkud zavádějící, jelikož se stále pohybujeme v prostředí infrastruktury společnosti Google. Nicméně tento pojem je zde uváděn právě kvůli existenci samostatného serveru pro vyřizování GCM požadavků, odděleného od aplikačního serveru, na němž je naše aplikace hostována. Následující *Tabulka 4* uvádí nejdůležitější parametry komunikace, které jsou v implementované aplikaci použity.

Důležité parametry GCM komunikace	
Sender ID	Toto ID je vygenerováno na základě hodnoty API key, kterou lze získat po přihlášení do vývojářské konzole. API key lze vygenerovat v sekci *APIs & auth*. Sender ID je použito při registračním procesu k identifikaci serveru třetí strany, který je určen pro zasílání zpráv na zařízení Android.
Application ID	Hodnota Application ID identifikuje danou aplikaci, která požaduje komunikaci GCM, tak aby mohly být GCM zprávy doručeny správné aplikaci. Application ID je odvozeno od názvu balíku uvedeného v souboru `AndroidManifest.xml`.
Registration ID	Registrační ID přidělené GCM serverem aplikaci na zařízení Android. Pomocí tohoto ID je jednoznačně identifikováno dané zařízení a konkrétní aplikace, která se podílí na komunikaci prostřednictvím GCM.

Tabulka 4. Důležité parametry GCM komunikace [27].

Pro použití GCM je nutné vygenerovat zmíněné API key, které slouží pro identifikaci oprávněného přístupu v případě, kdy aplikace zařízení Android žádá o přístup k některému z aktivních API, běžících na aplikačním serveru. Ve vývojářské konzoli je tedy nejprve nutné povolit GCM API a následně pro přístup k tomuto API vygenerovat API key. Obě tato nastavení lze provést v sekci *APIs & auth*. Povolení GCM provedeme v podsekci *APIs*, vyhledáním položky *Google Cloud Messaging for Android* a jejím přepnutím do pozice ON (viz *Obrázek 18*).



Obrázek 18. Povolení služby Google Cloud Messaging pro zařízení Android.

Druhý nezbytný krok, tedy vygenerování klíče API key, provedeme v podsekci *Credentials*. Zde kliknutím na tlačítko *Create new key* otevřeme dialogové okno, ve kterém vybereme možnost *Android key*. Následně je vyžadováno zadání řetězce, obsahujícího SHA1 otisk aplikace a název hlavního balíku. SHA1 otisk lze získat při exportování aplikace v prostředí Eclipse nebo přímo použitím nástroje keytool.exe, jak je uvedeno v dokumentaci na webu Google Developers[28], viz následující ukázka.

```
keytool -exportcert -alias key-name -keystore /file-path -list -v
```

Způsob použití GCM rozhraní na straně aplikačního serveru je opět uveden v krátké ukázce. Nejprve je vytvořena instance třídy `Sender`, která jako parametr přijímá výše zmíněné API key, vygenerované prostřednictvím konzole. Poté je vytvořena GCM zpráva. Zde je důležitá metoda addData(), která obsahuje název parametru, pod kterým budeme k obsahu zprávy přistupovat a následně její obsah. V našem případě se jedná vždy o krátké textové zprávy identifikující jednotlivé akce prováděné s daty v Google Datastore. Například při smazání projektu je odeslána zpráva obsahující řetězec „`PROJECT_DELETED`“. Na základě těchto zpráv jsou pak v klientské aplikaci vykonány patřičné operace (např. aktualizace seznamu projektů, apod.).

```
import com.google.android.gcm.server.Message;
import com.google.android.gcm.server.Result;
import com.google.android.gcm.server.Sender;
// ...
Sender sender = new Sender(API_KEY);

Message msg = new Message.Builder().addData("message", message).build();
Result result = sender.send(msg, deviceInfo.getDeviceRegistrationID(), 5);
```

[28]Web:https://developers.google.com/games/services/android/troubleshooting#check_the_certificate_fingerprint

6.2.7 Lokální vs. produkční server

Pro účely vývoje a testování, lze serverovou část App Engine aplikace spouštět i na lokálním počítači. V prostředí Eclipse k tomu není zapotřebí žádného zvláštního nastavení, kromě již dříve zmíněného *Google plugin*. Spuštění aplikace na lokálním serveru je realizováno přes nabídku projektu: *Run As > Web Application*. Po spuštění serveru je při výchozím nastavení dostupné webové rozhraní pro správu na adrese `http://localhost:8888/_ah/admin/`. Na této adrese je možné spravovat lokální datové úložiště, indexy, moduly a různá nastavení, která však pro nás nejsou až tak podstatná. Z pohledu implementované aplikace je v tuto chvíli nejdůležitější právě prohlížeč datového úložiště, pomocí kterého si můžeme ověřovat výsledky dotazů. Dalším užitečným nástrojem je prohlížeč dostupných API. Nachází se na adrese `http://localhost:8888/_ah/api/explorer/` a v tomto případě se jedná o seznam jednotlivých Cloud Endpoint rozhraní. Pro každou ze tříd `ProjectEndpoint.java`, `PortfolioEndpoint.java` a dalších, obsažených v projektu Google App Engine je zde uveden příslušný odkaz na jednotlivé metody těchto koncových bodů. Tímto způsobem je tedy možné testovat dané metody i bez nutnosti spuštění klientské aplikace, která těchto metod v běžném provozu používá.

Jsme-li s vývojem aplikace hotovi a plánujeme její přesun na produkční server, jsou na řadě následující velmi jednoduché kroky. Zaprvé je třeba mít správné nastavení projektu. Minimálně je nutné v nastavení projektu *Properties > Google > App Engine* nastavit hodnoty *Application ID* a *Version*. První z hodnot odpovídá proměnné *Project ID,* tedy názvu projektu, který byl vygenerován při vytvoření nové aplikace v Google Developers konzoli. Verze aplikace by měla být automaticky vyplněna. Pokud tomu tak není, zvolíme např. hodnotu 1. Na této proměnné až tak nezáleží. Ovlivňuje pouze konečnou adresu, kterou je daná aplikace identifikována na serveru Google. Pro publikování na produkční server pak stačí jen zvolit z nabídky projektu *Google > Deploy to App Engine*. Na následující obrazovce vybereme součásti aplikace určené k exportu na produkční server a potvrdíme stiskem tlačítka *Deploy*. Následuje proces nahrávání souborů aplikace na produkční server, jehož průběh je zobrazen v konzoli aplikace Eclipse. Po skončení tohoto procesu je uživatel informován o výsledku operace a v případě úspěšného nahrání je možné danou aplikaci ihned používat.

6.3 Klientská část

V této podkapitole budou zmíněny implementační detaily aplikace pro zařízení platformy Android. Strukturou projektu se zde již zabývat nebudeme, jelikož byla uvedena ve druhé kapitole této práce. Následuje popis uživatelského rozhraní a implementace vybraných částí aplikace.

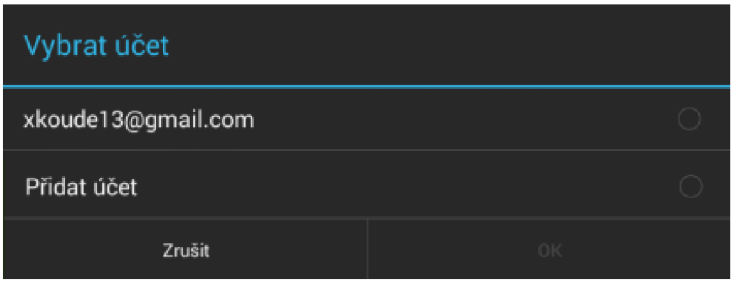
6.3.1 Grafické uživatelské rozhraní

Jak již bylo uvedeno v předchozích kapitolách a jak z názvu práce vyplývá, je implementovaná aplikace určena primárně pro dotykové tablety. Celkové rozložení jednotlivých grafických prvků je tedy organizováno do horizontálního zobrazení (*landscape*). Tomu jsou přizpůsobeny jednotlivé fragmenty, které jsou na obrazovkách aplikace děleny do dvou až tří sloupců. Aplikace se snaží zachovat kompatibilitu i pro jiné typy tabletů, než je testovací zařízení. Toho je dosaženo především použitím relativních hodnot u rozměrů prvků grafického rozhraní. Velikosti vykreslovaných prvků jsou uváděny v jednotkách *dp*[29], velikost písma je pak zadána v jednotkách *sp*[30]. Princip použití těchto jednotek je ten, že jsou založeny na hustotě bodů daného displeje. Tedy vyjadřují poměr počtu bodů na jeden skutečný pixel. Poměry pro různá rozlišení jsou dány následovně: ldpi (0.75), mdpi (1.0), hdpi (1.5), xhdpi (2.0). Na obrazovce s rozlišením mdpi tedy platí 1 dp = 1 px, na displeji s rozlišením xhdpi platí 1 dp = 2 px, atd.

6.3.1.1 Přihlašovací obrazovka

Přihlašovací obrazovka je vstupním bodem celé aplikace. Oproti původnímu návrhu, který byl zobrazen v kapitole 5.6, bylo učiněno několik změn. Tou první je celkový grafický styl aplikace, který je laděn do méně agresivní zelené barvy. Druhou změnou je pak způsob přihlašování. Původně bylo plánováno využít vlastní registrace a přihlašování, nicméně po hlubším rozboru byla vybrána možnost použití autentizace pomocí účtů Google (Google Account Picker).

Po kliknutí na tlačítko *Přihlásit*, se otevře dialogové okno s výběrem účtů, které jsou v daném zařízení dostupné. V případě, že není k dispozici žádný účet, je možné prostřednictvím tohoto dialogu vytvořit nový.

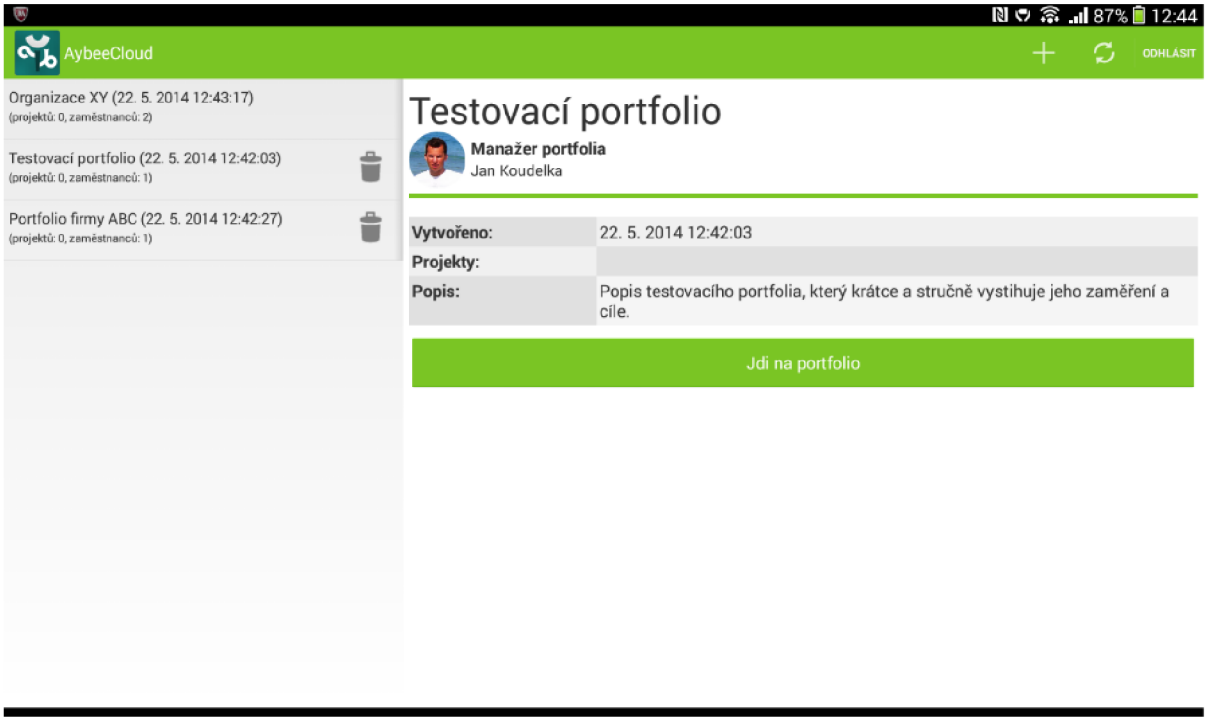


Obrázek 19. Dialog pro výběr účtu (Google Account Picker).

[29] Dp =Density-independent pixel (pixel nezávislý na hustotě bodů displeje).
[30] Sp = Scale-independent pixel (podobně jako u *dp*, slouží však k definici velikosti písma).

6.3.1.2 Přehled dostupných portfolií

Po výběru uživatelského účtu a úspěšném přihlášení se uživatel dostává na obrazovku, poskytující výběr portfolia a zobrazení základních informací o portfoliu, jako např. datum vytvoření, název, popis, informace o manažerovi portfolia, apod. V seznamu v levé části obrazovky jsou uvedena veškerá portfolia, ve kterých je uživatel jakýmkoliv způsobem angažován. Je-li přihlášený uživatel současně manažerem daného portfolia, pak má na rozdíl od ostatních právo vybrané portfolio odstranit. Výběrem položky v levém sloupci se zobrazí informace o portfoliu ve fragmentu na pravé straně obrazovky. Pod těmito údaji se pak nachází tlačítko pro přechod k vybranému portfoliu.



Obrázek 20. Obrazovka s přehledem dostupných portfolií.

Na této obrazovce lze rovněž vytvářet nová portfolia, a to kliknutím na ikonu se symbolem „+“ v horní liště (*action bar*) aplikace. Pro vytvoření portfolia je třeba zadat pouze název a krátký popis, který uvádí význam portfolia, případně jeho zaměření, cíle, apod. Po založení nového portfolia je jeho manažerem automaticky jmenován uživatel, který ho vytvořil. Toto nastavení je později možné změnit. V liště obsahující menu jsou dále ještě dvě položky. První z nich slouží pro znovunačtení seznamu portfolií. Při běžném provozu by toto tlačítko nemělo být zapotřebí, jelikož seznam je aktualizován při jakýchkoliv změnách (smazání portfolia, vytvoření nového, změna nastavení, atd.). Nicméně v ojedinělých případech (např. výpadek připojení k internetu) může být proces aktualizace přerušen. Právě v této chvíli je vhodné použít tlačítka pro znovunačtení. Poslední položkou v menu je tlačítko *Odhlásit*, které odhlásí aktuálního uživatele z aplikace a navrátí ho na úvodní přihlašovací obrazovku aplikace.

6.3.1.3 Obrazovka zobrazení portfolia

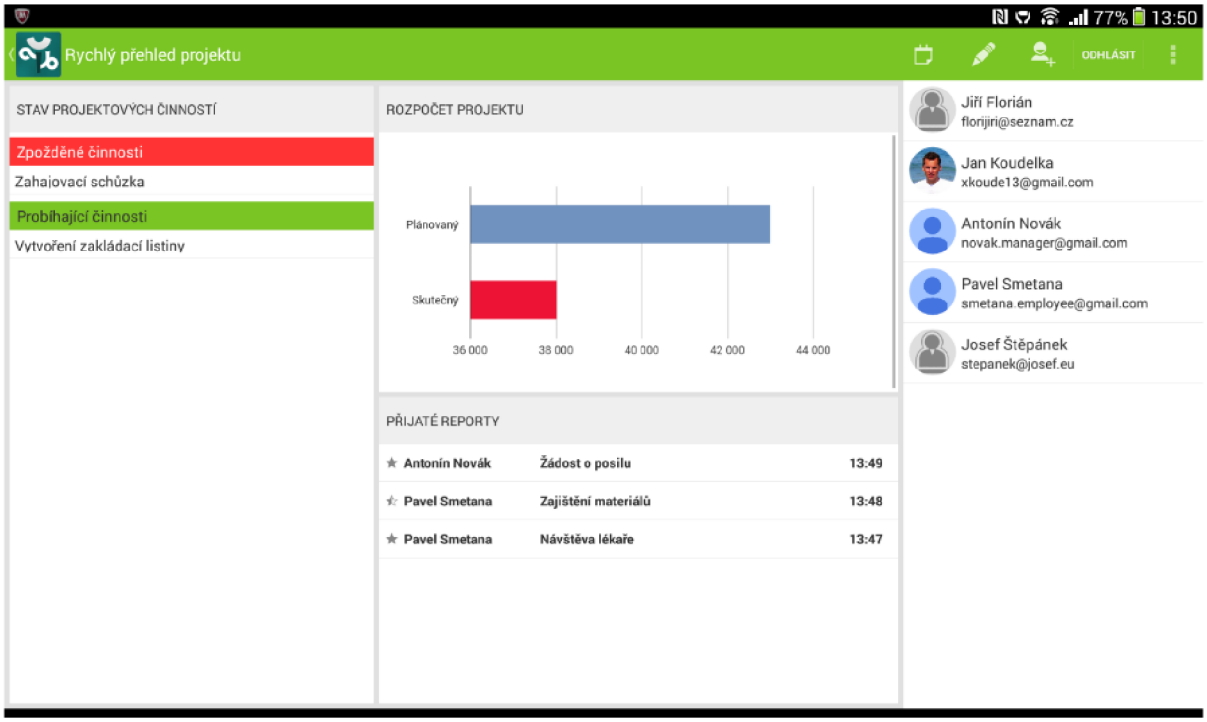
Po výběru portfolia se uživatel přesouvá na obrazovku, která slouží pro přehled o dostupných projektech. Dále je zde k dispozici fragment, obsahující jednoduchý komunikační nástroj ve stylu sociálních sítí. Tato komunikace je především určena pro věcné dotazy či připomínky týkající se celého daného portfolia, nikoliv pro frekventovanou komunikaci mezi dvěma či více uživateli.

Zobrazení možností této obrazovky opět závisí na oprávnění přihlášeného uživatele. Pokud je přihlášený zaměstnanec ve funkci manažera portfolia, má navíc k dispozici funkce přidávání resp. editace pracovníků portfolia, vytváření resp. mazání projektů, změna nastavení portfolia.

Pro ostatní role je možné z uvedených projektů pouze vybírat, přičemž jednotlivé položky jsou opět filtrovány podle zainteresovanosti přihlášeného uživatele. V seznamu projektů se tedy mohou nacházet položky, u kterých je uživatel veden jako manažer projektu nebo pouze jako běžný zaměstnanec (oproti původnímu návrhu byla pro jednoduchost eliminována role vedoucího fáze projektu).

6.3.1.4 Obrazovka přehledu projektu

Poté co uživatel vybere na předchozí obrazovce některý z uvedených projektů, je dle daného oprávnění přesměrován buď na rychlý přehled projektu (manažer projektu nebo manažer portfolia), anebo na obrazovku s výpisem obsažených činností (běžný zaměstnanec). Druhá zmíněná možnost pak poskytuje pro běžného zaměstnance přehled pouze těch činností v projektu, ke kterým je přiřazen. Obrazovka přehledu projektu je shodná pro manažera portfolia i manažera projektu s rozdílnými možnostmi editace.



Obrázek 21. Rychlý přehled projektu v režimu manažera projektu resp. manažera portfolia.

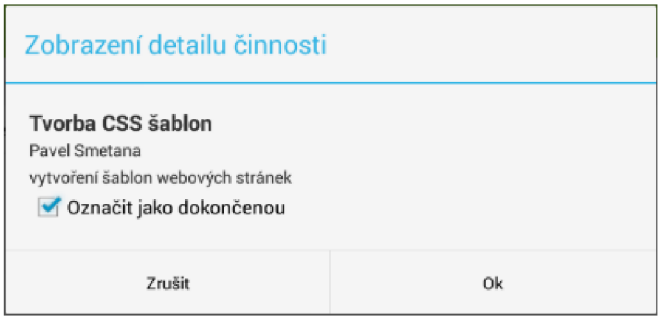
V levé části této obrazovky jsou uvedeny seznamy činností filtrovaných podle jejich aktuálního stavu. V případě, že je aktuální čas větší než koncový čas dané činnosti a dosud u ní nebyl změněn stav na *dokončená*, je tato činnost zařazena do seznamu zpožděných. Následuje seznam probíhajících, plánovaných a dokončených činností.

Prostřední část je rozdělena na dvě poloviny. Vrchní polovina obsahuje graf znázorňující odchylku skutečného rozpočtu od plánovaného. Plánovaný rozpočet je možné editovat v nastavení projektu. Skutečný rozpočet se počítá na základě definovaných činností, počtu pracovníků, jim přidělených rolí a hodinových sazeb. Druhá polovina této části je tvořena fragmentem, zobrazujícím přijaté reporty. Ty jsou rozlišeny na dva typy. Jedná se o reporty určené projektovému manažerovi nebo manažerovi portfolia. Reporty daného projektu jsou tedy filtrovány podle aktuálně přihlášeného uživatele a jeho pozice v projektu.

Zcela vpravo se dále nachází seznam všech zaměstnanců projektu, včetně jeho manažera. Na obrázku výše (*Obrázek 21*) lze vidět rozdílné zobrazení různých zaměstnanců. V tomto případě je reálným uživatelem pouze uživatel *Jan Koudelka*. Uživatelské účty *Antonín Novák* a *Pavel Smetana* jsou sice reálné účty na serveru Google, nicméně jedná se o fiktivní uživatelské účty vytvořené pro testovací účely. Aplikace však umožňuje i přidávání uživatelů, kteří ji prozatím nepoužívají, zadáním jejich emailové adresy, jména a příjmení. V případě, že se daný uživatel následně do aplikace přihlásí, jsou ze serveru Google staženy dodatečné informace o tomto uživateli, jako skutečné jméno uvedené v profilu, profilová fotografie, apod. Pokud se uživatel nepřihlásí, je v seznamu zobrazen se zašednutým profilovým obrázkem. Na výše uvedeném obrázku tedy můžeme vidět, že uživatelé *Jiří Florián* a *Josef Štěpánek* se doposud k aplikaci nepřihlásili.

6.3.1.5 Přehled činností běžného zaměstnance

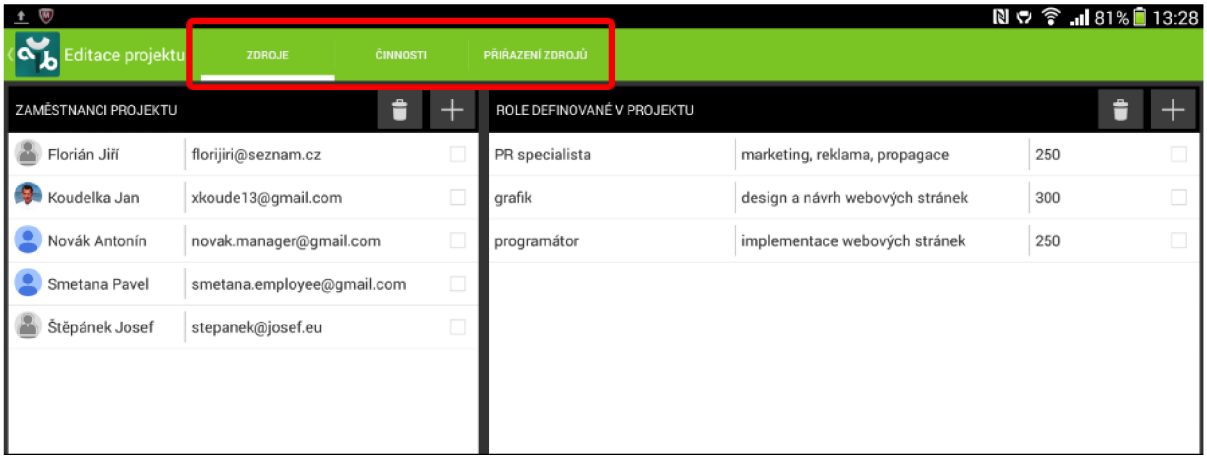
Výběrem projektu z pozice běžného pracovníka se uživatel dostává na seznam činností, které jsou mu přiřazeny. V tomto seznamu jsou činnosti řazeny stejně jako ve výše uvedeném přehledu projektu do kategorií podle jejich aktuálního stavu. Kliknutím na vybranou činnost se otevře dialogové okno, které poskytuje bližší popis dané činnosti a umožňuje pracovníkovi změnit stav činnosti po dokončení všech prací. Kromě toho může zaměstnanec výběrem příslušné položky v menu odesílat reporty manažerovi projektu.



Obrázek 22. Dialog zobrazení informací o činnosti s možností editace stavu.

6.3.1.6 Editace projektu – definice rolí

Jednotliví pracovníci projektu jsou ve vztahu k činnostem v určitých rolích. Tyto role je možné definovat po kliknutí na položku *Upravit projekt* v menu přehledové obrazovky projektu. Tím se zobrazení přepne na editační obrazovku, která se skládá ze tří záložek (viz *Obrázek 23*). První z nich je editace zdrojů. Zde je možné spravovat pracovníky projektu a role definované projektem. Definice role obsahuje název, krátký popis a hodinovou sazbu pro výpočet mzdy, respektive skutečného rozpočtu.



Obrázek 23. Obrazovka pro správu projektu.

6.3.1.7 Editace projektu – definice činností

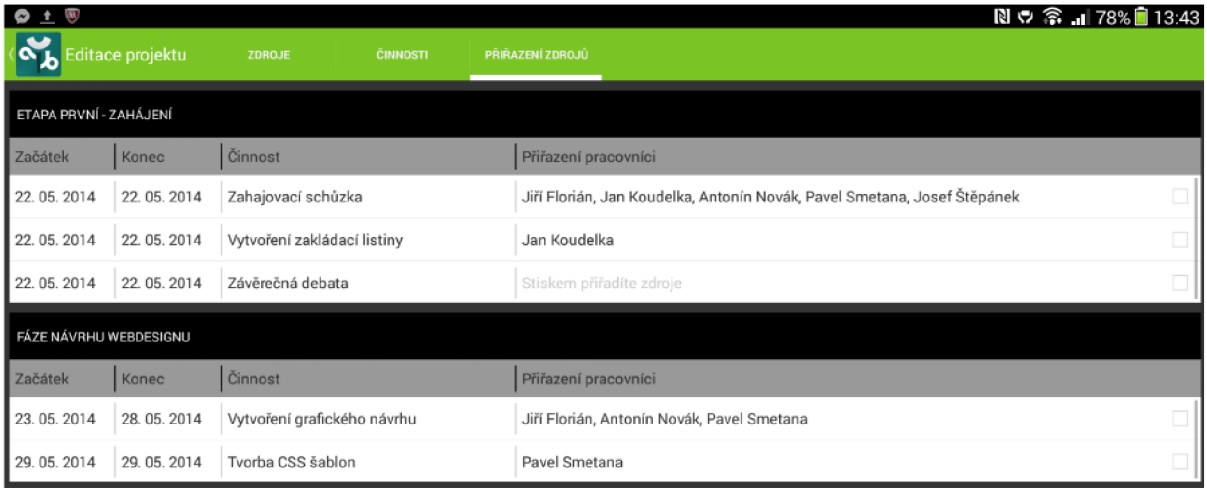
Další záložkou na výše zmíněné obrazovce jsou *Činnosti*. Zde podobně jako u zdrojů je možné vytvářet jednotlivé položky hierarchie činností (WBS – viz kapitola 3.2.1), které lze organizovat do jednotlivých fází a v rámci těchto fází do konkrétních činností.

6.3.1.8 Editace projektu – přiřazení zdrojů

Z hlediska managementu projektu je tato třetí kategorie správy projektu nejzásadnější. Jde o sekci určenou k mapování zdrojů na definované činnosti. Na této obrazovce je uveden seznam všech fází projektu a k nim příslušejících činností. Každý řádek zde reprezentuje jednu činnost, přičemž platí vzestupné uspořádání podle času zahájení činnosti. Nové přiřazení lze provést klepnutím na vybranou činnost. Tím se otevře dialog pro správu přiřazených zdrojů, podobný tomu na *Obrázek 24*.

Obrázek 24. Dialog pro správu přiřazení zdrojů k činnostem.

Dialog uvedený na předchozím obrázku zobrazuje aktuální mapování zdrojů na činnosti. Každý řádek se skládá ze dvojice *zaměstnanec – role*, přičemž u obou parametrů je možné vybírat z definovaných entit prostřednictvím výběrového menu (tzv. *spinner*). Pro přidání nového přiřazení slouží tlačítko *Přidat*, které dynamicky vytváří v aktivním dialogu nové řádky pro editaci. Jakékoliv z uvedených přiřazení lze jednoduše odebrat stisknutím tlačítka se symbolem křížku - vpravo na konci každého řádku. Provedené změny je třeba uložit stisknutím tlačítka *Použít*. Naopak pokud nechceme změny ukládat lze dialog stisknutím tlačítka *Zrušit* jednoduše ukončit bez zásahu do projektu. Příklad obrazovky po přiřazení zdrojů je uveden na *Obrázek 25*.



Obrázek 25. Obrazovka přiřazení zdrojů k projektovým činnostem.

6.3.2 Funkční logika aplikace

Celkově se aplikace neliší žádným zásadním způsobem od ostatních aplikací pro Android, tedy alespoň co se týče použitých programovacích technik a principů. Využívá standardních prvků grafického rozhraní. Jednotlivé obrazovky jsou reprezentovány formou tříd rozšiřujících třídu `android.app.Activity`. V rámci těchto tříd jsou většinou dále děleny na dva, či více oddílů třídy `android.app.Fragment`, případně `android.support.v4.Fragment`. Dále je pro navigaci použito prvku *Action Bar*. Tento prvek, který se zobrazuje jako lišta v horní části obrazovky, současně slouží pro identifikaci aktuální obrazovky formou titulku, či loga aplikace. Kromě toho může obsahovat záložky a také položky kontextového menu.

Všechny tyto prvky jsou běžně používány v mnoha různých aplikacích. To, co je typické pro aplikace našeho typu, jsou techniky používané buď speciálně v aplikacích na platformě Google App Engine nebo v aplikacích řešících obecně problematiku cloudových technologií. Dále jsou uvedeny některé zajímavé součásti aplikace z hlediska implementace její funkčnosti.

6.3.2.1 Asynchronní úlohy

Asynchronní úlohy jsou jedním z principů, který je v implementované aplikaci velmi často používán ve spojitosti se zpracováním požadavků na datové úložiště. Základní princip spočívá v tom, že data v rámci asynchronní úlohy jsou zpracovávána v samostatném vlákně programu, nikoliv ve vlákně uživatelského rozhraní. Předchází se tak nežádoucímu zamrznutí aplikace, které by jinak při vysoké náročnosti daných operací mohlo trvat i několik desítek vteřin či déle. Kromě přihlašovacího procesu jsou asynchronní úlohy použity právě pro dotazy nad datovým úložištěm Google Cloud Datastore, pomocí klientských koncových bodů Google Cloud Endpoints. Typický příklad použití pro tyto účely je uveden v ukázce níže.

```
private class GetUsersAsyncTask
       extends AsyncTask<Void, Void, CollectionResponseUser> {

   @Override
   protected void onPreExecute() {
      super.onPreExecute();
      // ...
   }

   @Override
   protected CollectionResponseUser doInBackground(Void... params) {
      CollectionResponseUser response = null;
      try {
         Userendpoint.Builder endpointBuilder = new Userendpoint.Builder(
            AndroidHttp.newCompatibleTransport(), new JacksonFactory(),
            new HttpRequestInitializer() {
               public void initialize(HttpRequest httpRequest) {
               }
            });

         userEndpoint =
               CloudEndpointUtils.updateBuilder(endpointBuilder).build();

         response = userEndpoint.listUser().execute();
      } catch (Exception e) {
         Log.d("Could not get user list", e.getMessage(), e);
      }
      return response;
}
```

Jak lze vidět, úloha obsahuje implementaci několika přetížených metod (`@Override`), přičemž postačujícím základem je metoda `doInBackground()`. V rámci ní jsou realizovány zvolené asynchronní operace. Název ostatních dvou metod pak vypovídá o jejich použití. Metoda `onPreExecute()` je volána před provedením `doInBackground()` a metoda `onPostExecute()` naopak po provedení asynchronních operací. Těchto metod lze například využít pro upozornění uživatele o průběhu dané úlohy. V těle metody `doInBackground()` na výše uvedené ukázce můžeme vidět způsob vytvoření požadavku na datové úložiště. V tomto případě je použita metoda

`listUser()` nad koncovým bodem *Userendpoint*, který obstarává dotazy nad entitou typu *User*. Proměnná `userEndpoint` je zde typu `Userendpoint`, dle odpovídající třídy koncového bodu `UserEndpoint.java`. Při změně implementace pro dotazy nad jiným typem entity se změní právě tato proměnná a dále se analogicky změní i název třídy u proměnné `endpointBuilder`.

6.3.2.2 Broadcast receiver pro reakce na GCM zprávy

Formát zprávy, odesílané prostřednictvím Google Cloud Messaging (GCM) byl popsán v předchozí kapitole věnující se serverové části aplikace. Data zde byla prostřednictvím metody `addData()` zabalena do výsledné zprávy a odeslána přes GCM server zaregistrovaným zařízením. V klientské aplikaci je proces registrace k GCM proveden během přihlašování. V rámci této registrace je vytvořena instance třídy `GCMIntentService.java`, která běží na pozadí celé aplikace jako služba a zpracovává příchozí zprávy od GCM serveru. Po přijetí zprávy je následně volána metoda `onMessage()`, obsahující implementaci odpovídajících reakcí na základě typu přijaté zprávy. Kód obsažený v této metodě se však stará pouze o rozlišení zpráv a jejich přeposlání metodou `sendBroadcast()` s odpovídajícím parametrem. Pro příklad uveďme reakci na zprávu o vytvoření nového portfolia.

```
Intent in = new Intent();
in.setAction("com.google.xkoude13.aybeecloud.portfolio.added");
sendBroadcast(in);
```

V ukázce vidíme, že byl vytvořen `Intent` a nad ním volána metoda `setAction()` s textovým řetězcem, jenž by měl být v rámci aplikace unikátní pro každou jednotlivou operaci, kterou chceme tímto způsobem zpracovávat. Dále je zapotřebí pro tuto zprávu implementovat odpovídající *broadcast receiver*. Jelikož při vytvoření nového portfolia chceme aktualizovat seznam položek na obrazovce s výběrem portfolií, bude implementace *broadcast receiveru* realizována v rámci odpovídající třídy *Activity* (v tomto případě `PortfolioActivity.java`).

```
private class PortfolioReceiver extends BroadcastReceiver {

   @Override
   public void onReceive(Context context, Intent intent) {

      String action = intent.getAction();
      if(action.equals("com.google.xkoude13.aybeecloud.portfolio.added"))
         // reload list of the portfolio items
   }
}
```

Pro fungování takto vytvořeného *broadcast receiveru* je nutné ho pro danou třídu *Activity* nejprve zaregistrovat pomocí metody `registerReceiver()`. Vhodný okamžik pro volání této metody je v průběhu vykonávání `onResume()`, která je volána při aktivaci dané aktivity. Naopak

při volání `onPause()` je důležité zaregistrovaný *broadcast receiver* odregistrovat pomocí metody `uregisterReceiver()`. Na ukázce níže lze vidět způsob registrace s parametrem odpovídajícím výše zmíněnému typu zprávy. Pro zpracování více typů zpráv tímto receiverem pak stačí pouze přidat další řádky typu `intentFilter.addAction("identifikátor zprávy")`.

```
IntentFilter intentFilter = new IntentFilter();
intentFilter.addAction("com.google.xkoude13.aybeecloud.portfolio.added");
registerReceiver(new PortfolioReceiver(), intentFilter);
```

6.3.2.3 Notifikace

Notifikace určitým způsobem navazují na předchozí zpracování zpráv, jelikož jsou použity pro upozornění uživatele právě na změny způsobené přijetím těchto zpráv. Kromě aktualizace uživatelského rozhraní se tedy při přijetí Google Cloud Message objeví nový záznam v notifikační liště zařízení upozorňující na provedené změny. Zobrazení notifikací je tedy realizováno standardním způsobem dle dokumentace.

6.3.2.4 Implementace seznamů pomocí adaptérů

Jedním ze způsobů jak prezentovat větší množství dat stejného typu může být použití tabulek. Nicméně v prostředí platformy Android je tento princip poměrně nešikovný a dokonce není ani příliš doporučovaný. Lepším způsobem je tedy využití prvku *ListView* jedná-li se o seznam, který je součástí layoutu obsahujícího další grafické prvky. Případnou další možností je použití fragmentů rozšiřujících třídu `android.app.ListFragment`. Obě zmíněné metody poskytují základní rozhraní pro vytváření adaptérů k naplnění seznamu daty, nicméně pro definici vlastního vzhledu a určení dat, která mají být v rámci jednotlivých položek zobrazena, je nutné použít vlastních adaptérů. Jedním ze způsobů implementace vlastního adaptéru je rozšíření třídy `android.widget.ArrayAdapter<…>`. Takto vytvořený adaptér pak obsahuje pouze konstruktor a minimálně jednu přetíženou metodu (`@Override`) `getView()`, která definuje vzhled a zobrazená data pro jednotlivé položky seznamu na základě pozice určené parametrem `position`.

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {…}
```

Vytvoření seznamu pak probíhá v následujících krocích. Nejprve jsou z datového úložiště získány potřebné záznamy, které mají být zobrazeny. Následně je vytvořena instance třídy vlastního definovaného adaptéru. Posledním krokem je nastavení takto vytvořeného adaptéru prvku grafického rozhraní. Tedy v případě použití *ListView* se je nad odpovídajícím objektem volána metoda `setAdapter()` s parametrem objektu vytvořeného adaptéru. Při použití *ListFragment* je situace

podobná, s tím rozdílem, že metoda se nazývá `setListAdapter()`. Parametry jsou u obou metod stejné.

6.3.2.5 Navigace napříč aplikací

Kromě navigace v rámci portfolia, či projektu, která je realizována pomocí nabídek menu v *Action Bar* liště, je uplatněna i navigace v rámci hierarchie jednotlivých obrazovek aplikace. Tedy například při přechodu z obrazovky s přehledem portfolií na detail vybraného portfolia, se lze opět navrátit k předchozímu přehledu stisknutím tzv. *home* tlačítka umístěného vlevo na horní liště aplikace. Tento způsob navigace vyžaduje nejprve nastavení zobrazení a povolení zmíněného tlačítka v každé jednotlivé třídě *Activity*, ve které je toto chování požadováno. Nastavení lze provést v metodě `onCreate()`, přidáním následujících řádků.

```
ActionBar actionBar = getActionBar();
actionBar.setDisplayOptions(ActionBar.DISPLAY_SHOW_HOME);
actionBar.setDisplayHomeAsUpEnabled(true);
```

Pro navigaci tímto způsobem je dále třeba definovat hierarchii jednotlivých tříd, aby bylo jasné, mezi jakými třídami se má navigovat. Není nutnou podmínkou, že nadřazená třída *Activity* musí v průchodu aplikací přímo předcházet třídě aktuální. Definice této hierarchie se uvádí v souboru `AndroidManifest.xml` a její struktura vypadá následovně.

```
<activity
  android:name="com.google.xkoude13.aybeecloud.portfolio.PortfolioActivity"/>

<!-- A child of the portfolio activity -->
<activity
  android:name="com.google.xkoude13.aybeecloud.portfolio.AddPortfolioActivity"
  android:label="@string/portfolio_add_title"
  android:parentActivityName="com.google.xkoude13.aybeecloud.portfolio
   .PortfolioActivity"/>
```

Implementace chování po stisknutí tlačítka *home* je stejně jako u všech položek menu realizována v metodě `onOptionsItemSelected()` a obsahuje pouze jediný řádek (viz následující úryvek zdrojového kódu).

```
NavUtils.navigateUpFromSameTask(this);
```

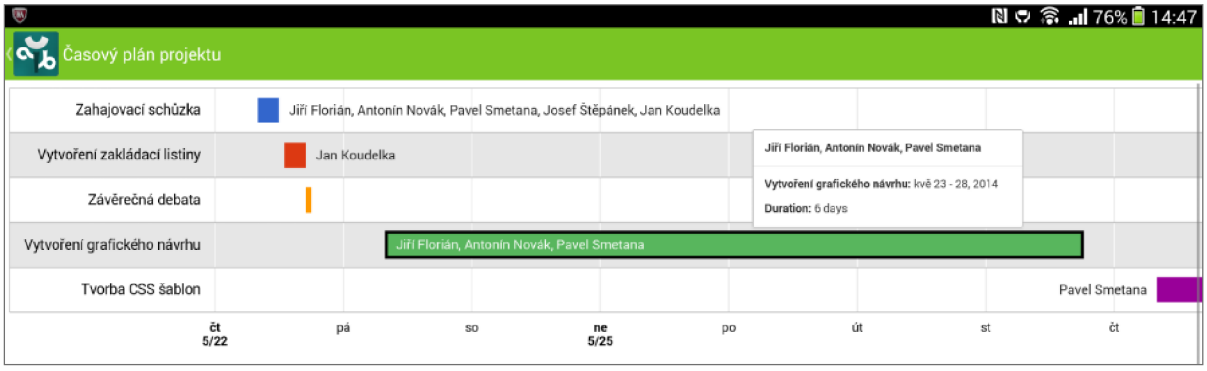
6.4 Další použitá API

Kromě výše uvedených API, které jsou přímo součástí platformy Google Cloud, byla použita dále některá aplikační rozhraní třetích stran. Použití těchto API je v souladu s podmínkami jejich distribuce a používání.

6.4.1 Google Charts API

V aplikaci je v několika případech pro zobrazení dat statistické povahy použito rozhraní Google Charts. V tomto případě se tedy nejedná až tak doslova o třetí stranu, nicméně vzhledem k tomu, že toto rozhraní není přímo součástí platformy Google Cloud, můžeme ho za API třetí strany považovat. Jeho použití je jako většina služeb společnosti Google zdarma. Rozhraní slouží pro zobrazení dat formou grafů různých typů. Přehled všech druhů poskytovaných grafů je možné nalézt na webových stránkách, které dokumentují použití tohoto API[31]. Aplikace využívá pouze některé vybrané typy výše zmíněných grafů. Jedná se o tyto dva:

- Bar Chart (sloupcový) – pro zobrazení skutečného stavu rozpočtu oproti plánovanému
- Timeline (časová osa) – pro zobrazení jednoduchého Ganttova diagramu



Obrázek 26. Příklad použití grafu pro zobrazení časového plánu (Ganttův diagram).

Způsob, jakým jsou grafy v mobilní aplikaci generovány, je zcela jednoduchý. Pro načtení takového grafu je použito klasické html stránky, která obsahuje kód v jazyce Javascript, jenž se stará o vygenerování grafu. Následně je tato stránka načtena prostřednictvím prvku WebView grafického rozhraní Android do předem určeného fragmentu aplikace.

6.4.2 DateTime Picker API

Dalším použitým API je rozhraní DateTime Picker. Slouží pro výběr data respektive času, přičemž jde o věrnou napodobeninu výběrového dialogu z prostředí mobilních aplikací Google. Toto API je

[31] Web: https://developers.google.com/chart/interactive/docs/gallery?hl=cs.

šířené pod licencí Apache License 2.0[32], a je ke stažení na serveru GitHub[33]. Vzhled výběrového dialogu je zobrazen na obrázku níže. Tohoto dialogu je použito při vytváření záznamů obsahujících časový údaj zahájení a ukončení. Jde tedy především o využití na obrazovkách pro vytváření projektu, editaci projektu, vytvoření a editace fáze projektu, vytvoření a editace činnosti v projektu a v dalších podobných situacích.

Obrázek 27. Dialog pro výběr data využívající DateTime Picker API.

6.5 Metriky aplikace

Následující tabulka uvádí pohled na základní metriky aplikace z různých pohledů, především se zaměřením na kód a výstupy aplikace. Údaje jsou rozděleny na serverovou část a klientskou část. Typickým údajem je počet řádků kódu, dále počet tříd, balíků a celková velikost výsledné aplikace. Z celkového počtu řádků v souborech s příponou `java` je vidět, že zhruba polovina je obsažena v generovaných souborech, případně v připojených knihovnách..

Počet řádků kódu v souborech `*.java` adresáře `src` klientské aplikace	13 202 řádků
Počet řádků kódu v souborech `*.java` adresáře `src` serverové aplikace	4 765 řádků
Celkový počet řádků kódu v souborech `*.java` klientské aplikace	35 089 řádků
Celkový počet řádků kódu v souborech `*.java` serverové aplikace	4 765 řádků
Celkový počet řádků kódu v `*.java` souborech (včetně generovaných)	39 854 řádků
Počet balíků adresáře `src` klientské aplikace	8 balíků
Počet tříd v adresáři `src` klientské aplikace	68 tříd
Velikost výsledného instalačního souboru `AybeeCloud.apk` klientské aplikace	2 036 kB

Tabulka 5. Metriky serverové a klientské části aplikace.

[32] Web: http://www.apache.org/licenses/LICENSE-2.0.
[33] Web: https://github.com/flavienlaurent/datetimepicker.

7 Testování

Následující část práce je věnována popisu testování mobilní aplikace navržené a implementované dle předchozích kapitol. Účelem je otestovat celkovou funkčnost aplikace, použitelnost jednotlivých komponent, množství spotřebovaných zdrojů hostitelského serveru a kompatibilitu s různými typy zařízení. Dalším neméně důležitým úkolem testování je zjištění případných chyb a jejich vyladění před vytvořením finální verze aplikace.

7.1 Testování kompatibility

Účelem testování kompatibility bylo ověření správného provozu aplikace na různých typech zařízení. V rámci tohoto procesu byla aplikace otestována především z hlediska zobrazovacích možností a celkové podpory aplikace na operačním systému daného zařízení. Pro tyto testy byly k dispozici tablety, které uvádí *Tabulka 6*. Parametry displeje jsou uváděny v jednotkách palců (úhlopříčka displeje), resp. v jednotkách pixelů (rozlišení displeje). Další parametry týkající se výkonu nejsou uvedeny, protože nejsou pro tento typ aplikace nijak zásadní – nejedná se o graficky či výpočetně náročnou aplikaci.

Název zařízení	Verze OS	Úhlopříčka	Rozlišení displeje
Sony Xperia Tablet Z	4.4.2 (Kit Kat)	10.1“	1920 x 1200
Samsung Galaxy Tab 3	4.1.2 (Jelly Bean)	7“	1024 x 600
Asus Memo Pad HD 7	4.2.2 (Jelly Bean)	7“	1280 x 800

Tabulka 6. Seznam reálných testovacích zařízení.

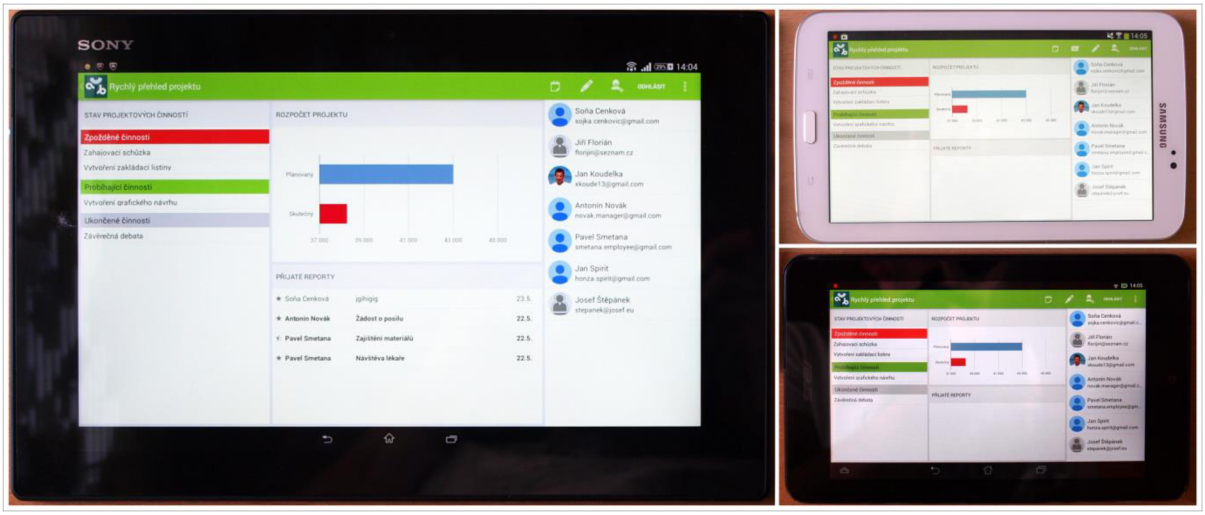
Primárním testovacím zařízením byl vlastní tablet Sony Xperia Tablet Z, na kterém byla aplikace průběžně testována pomocí režimu ladění přes rozhraní USB. U tohoto modelu se tedy předpokládala a také byla průběžně ověřována funkčnost aplikace již od počátečních fází vývoje. Rovněž co se týče grafického rozvržení, bylo optimalizováno právě pro tento tablet.

Zbývající dvě zařízení byla pro testovací účely zapůjčena. Instalace a spuštění proběhly opět v pořádku, bez jakýchkoliv chyb. Dle údajů z předchozí tabulky tedy můžeme předpokládat kompatibilitu tabletů s operačním systémem Android ve verzích 4.1.x - JellyBean a 4.2.x – JellyBean. Jelikož současná nejvyšší verze OS Android je verze 4.4 – KitKat, můžeme vzhledem k údajům prvního zmíněného tabletu považovat aplikaci za kompatibilní i s nejnovějším operačním systémem Android.

Zobrazení grafického rozhraní a rozmístění jednotlivých prvků aplikace bylo rovněž v pořádku, čehož bylo docíleno především díky použití relativních velikostí daných prvků a použitého písma. Pro bitmapové soubory pak byly použity různé verze obrazových dat v několika rozdílných

velikostech, rozřazené do podsložek adresáře `drawable`. Ten obsahuje podadresáře s prefixem `drawable` a koncovkou `-mdpi`, `-hdpi`, `-xhdpi`, `-xxhdpi`, odpovídající příslušným druhům displejů.

Kompatibilita s nižšími verzemi operačního systému nemohla být ověřena, jelikož nebylo k dispozici žádné reálné zařízení a použití softwarového emulátoru nebylo možné z důvodu příliš velkých nároků na systémové zdroje použitého počítače, na němž byla aplikace vyvíjena. Přesto by měla být kompatibilní minimálně se všemi verzemi Android od 4.0 výše a teoreticky (díky použití knihoven `support`[34]) i s nižšími až po verzi 3.0 – Honeycomb.



Obrázek 28. Fotografie reálného zobrazení aplikace na výše uvedených zařízeních.

7.2 Funkční testování

Předmětem funkčního testování bylo především ověřit správnou funkci všech prováděných operací v aplikaci. To znamená od vytvoření portfolia, přes výběr, přidávání pracovníků, vytváření projektů, správu a nastavení, až k jednotlivým dialogovým oknům, tlačítkům, atd. Cílem bylo odhalit chyby, které se neprojevily při průběžném testování během vývoje aplikace. Při testovacím provozu bylo zjištěno několik nekorektně ošetřených výjimek, které se projevovaly spíše sporadicky, avšak způsobovaly selhání aplikace. Byly ošetřeny například nestandardní situace týkající se přihlašování a síťového připojení.

Dále byla testována synchronizace mezi více zařízeními, kdy bylo opět odhaleno několik chyb. Tyto chyby způsobovaly nesprávné chování při aktualizaci dat o portfoliu. Z tohoto důvodu docházelo při úpravách na různých zařízeních k nekonzistencím v datovém úložišti. Revizí kódu se zjistily chybějící podmínky ve třídě `GCMIntentService.java`, které měly implementovat správnou aktualizaci dat portfolia. Dalším slabé místo aplikace bylo odhaleno v situaci, kdy uživatel

[34] Support libraries – knihovny pro podporu zpětné kompatibility, viz http://developer.android.com/tools/support-library/index.html.

přihlášený jako manažer odstranil portfolio z databáze, přičemž jiný uživatel prováděl operace v tomto portfoliu. Při jakékoliv následující operaci zaměstnance portfolia pak následovalo selhání aplikace. Situace byla ošetřena přidáním varování na straně uživatele, který požaduje odstranění, a zobrazením upozornění o smazání portfolia a následným přesměrováním na úvodní obrazovku na straně přihlášeného zaměstnance.

7.3 Testování výkonu

Aplikace byla původně implementována takovým způsobem, že při každé operaci týkající se načítání či ukládání dat z datového úložiště docházelo ke znovunačtení všech odpovídajících entit zobrazených na dané obrazovce aplikace. Tento způsob byl v zásadě relativně rychlý, nicméně co se týče množství I/O operací v datovém úložišti, bylo brzy dosaženo limitů pro poskytování služeb zdarma. Z tohoto důvodu bylo nutné aplikaci upravit tak, aby vždy aktualizovala pouze data, u kterých jsou prováděny nějaké změny. Aplikace si tak udržuje pro vybrané portfolio a pro konkrétní projekt veškeré informace načtené z datového úložiště. Při změně parametrů, přidání činností nebo jiných operacích, provede aplikace odpovídající operace nad datovým úložištěm a v případě úspěšného provedení změn aktualizuje i lokální kopii projektu udržovanou v kontextu běžící aplikace. Tímto způsobem bylo ušetřeno hlavně operací čtení, kterých při vytvoření pěti různých portfolií obsahujících několik projektů a zaměstnanců bylo spotřebováno během několika minut cca 20 % z celkového limitu. Provedením výše zmíněných úprav bylo dosaženo značných úspor. Následující obrázek znázorňuje část tabulky *Quota Details* pocházející z vývojářské konzole Google Developers. Při stejném počtu operací (vytvořených portfolií, projektů, zaměstnanců) byl počet operací čtení snížen z původních 20 % na 3 %.

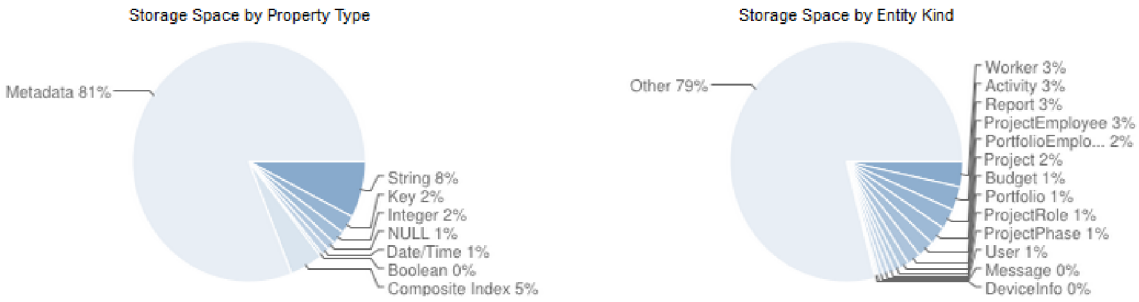
Obrázek 29. Ukázka ze stránky přehledu využití zdrojů.

Jiným parametrem zkoumání byla odezva aplikace při dotazech na Google Datastore. V případě použití lokálního testovacího serveru byla dle očekávání odezva velmi rychlá. Pouze prvotní inicializace lokálního datového úložiště někdy trvala příliš dlouho a často docházelo k vypršení časového limitu požadavku. S přechodem aplikace na produkční server se rychlost odezvy u některých operací nepatrně snížila, avšak u jiných bylo dosaženo dokonce lepších časů než na lokálním stroji. V zásadě se však nejedná o velké rozdíly. V obou případech jde maximálně o jednotky vteřin (u větších objemů dat), většinou však o zlomek sekundy (změna parametrů, zaslání reportu, apod.).

Do jisté míry může rychlost komunikace mezi serverem a aplikací ovlivňovat i geografická pozice datových serverů, mezi nimiž jsou data distribuována. Nastavení geopolitické lokalizace dat

podle požadavků uživatele je možné pouze v režimu prémiových účtů Google. Tento faktor však bude výslednou rychlost ovlivňovat zřejmě pouze v řádech milisekund.

Zajímavým údajem o využití datového úložiště Google Cloud Datastore může být následující obrázek s grafy vyjadřujícími rozdělení datového prostoru podle typu uložených hodnot a podle druhu entit. Samozřejmě, že tyto hodnoty jsou závislé na velikosti existujících portfolií, počtu projektů, apod., nicméně nám mohou tato data posloužit jako informace o průměrném využití datového úložiště.



Obrázek 30. Grafy využití datového prostoru Google Cloud Datastore.

Z uvedených grafů lze vyčíst, že podstatnou část uložených dat tvoří metadata. Jedná se především o informace týkající se druhů entit, datových typů vlastností jednotlivých entit, informace o indexech, relacích mezi entitami, apod.

8 Závěr

V rámci této práce jsem se seznámil s postupy a metodami tvorby aplikací pro mobilní platformu Android. Studiem souvisejících témat jsem získal velké množství nových informací, které považuji za užitečné z hlediska osobního vzdělání a pro účast na dalších projektech. Vzhledem ke stále rostoucímu počtu mobilních zařízení považuji tyto znalosti za velmi cenné.

Základem této práce bylo prostudovat tři hlavní oblasti, na které se zaměřuje. Těmito oblastmi jsou vývoj aplikací pro operační systém Android, problematika a způsob využití cloudových technologií a dále nástroje a používané techniky v projektovém řízení.

První oblasti se věnuje druhá kapitola této práce, která je zaměřena na obecný popis a historii operačního systému Android a dále pak na způsob vývoje aplikací pro tuto platformu. Poté následuje kapitola o projektovém managementu, obsahující definice základních pojmů používaných v projektovém řízení a popis některých standardně používaných nástrojů. Poslední teoretickou částí je čtvrtá kapitola, která pojednává o problematice cloudových technologií. Uvádí srovnání různých existujících řešení a porovnává výhody a nevýhody použití cloudu. Nakonec vysvětluje výběr technologií použitých v této práci. Pátá kapitola je věnována popisu návrhu aplikace. Obsahuje specifikaci požadavků, definici rolí, funkcí, apod. Následuje soubor návrhových diagramů pro popis aplikace, včetně diagramu struktury datového úložiště.

Během prvního semestru byl v rámci semestrálního projektu připraven návrh aplikace pro správu projektů, běžící na tabletech platformy Android a komunikující se serverovou částí v cloudu Google App Engine. Tento projekt dal teoretický základ pro další pokračování v rámci diplomové práce. Následovala samotná implementace navržené aplikace a poté její testování na reálných zařízeních. Implementace a testování jsou popsány v šesté a sedmé kapitole práce. Jako implementační jazyk byl použit jazyk Java a pro vývoj prostředí Eclipse v balíčku Android Developer Tools.

Navrženou aplikaci se podařilo s drobnými změnami realizovat a uvést do provozu. Byla úspěšně otestována na několika reálných zařízeních. Dalšími možnými kroky, navazujícími na tuto práci by mohla být lepší optimalizace spotřebovávaných zdrojů, případně rozšíření aplikace o desktopovou či webovou verzi. Širšímu využití by zcela jistě přispěla integrace s jinými nástroji pro projektové řízení, jako je např. MS Project. V rámci aplikace by rovněž mohl být implementován mechanismus importu resp. exportu dat. Bude-li to časově možné, bude v následujícím čtvrtletí připravena verze aplikace vhodná pro zveřejnění v Google Play.

Literatura

[1] Dashboards. *Android Developers* [online]. 2013 [cit. 2013-12-10]. Dostupné z: http://developer.android.com/about/dashboards/index.html.

[2] New in Android. *Android Developers* [online]. 2013 [cit. 2013-12-10]. Dostupné z: http://developer.android.com/design/patterns/new.html.

[3] GARGENTA, Marko. *Learning Android*. 1st ed. Sebastopol, CA: O'Reilly Media, 2011, xvii, 245 p. ISBN 978-1-449-39050-1.

[4] MATTHEWS, Robbie. *Beginning Android tablet programming starting with Android Honeycomb for tablets* [online]. Berkeley, CA: Apress, 2011 [cit. 2013-12-10]. ISBN 978-1-4302-3784-6. Dostupné z: http://it-ebooks.info/book/1308/.

[5] MEDNIEKS, Zigurd, Laird DORNIN, G. Blake MEIKE a Masumi NAKAMURA. *Programming Android*. 1st ed. Sebastopol: O'Reilly, 2011, xvi, 482 s. ISBN 978-1-4493-8969-7.

[6] DOUDĚRA, Martin. *Vizualizace Ganttova diagramu pro platformu Android*. Brno, 2013. Dostupné z: http://www.fit.vutbr.cz/study/DP/BP.php?id=15577&file=t. Bakalářská práce. FIT VUT v Brně.

[7] Application Fundamentals. *Android Developers* [online]. 2013 [cit. 2013-12-27]. Dostupné z: http://developer.android.com/guide/components/fundamentals.html.

[8] Starting an Activity. *Android Developers* [online]. 2013 [cit. 2014-01-02]. Dostupné z: http://developer.android.com/training/basics/activity-lifecycle/starting.html.

[9] SVOZILOVÁ, Alena. *Projektový management*. 1. vyd. Praha: Grada, 2006, 353 s. ISBN 80-247-1501-5.

[10] DOLEŽAL, Jan, Pavel MÁCHAL a Branislav LACKO. *Projektový management podle IPMA*. 2., aktualiz. a dopl. vyd. Praha: Grada, 2012, 526 s. Expert (Grada). ISBN 978-80-247-4275-5.

[11] ROSENAU, Milton D. *Řízení projektů: příprava a plánování, zahájení, výběr lidí a jejich řízení, kontrola a změny, vyhodnocení a ukončení*. Vyd. 2. Brno: Computer Press, 2003, xii, 344 s. ISBN 80-722-6218-1.

[12] BIAFORE, Bonnie. *Successful project management: applying best practices and real-world techniques with Microsoft Project*. Sebastopol, CA: O'Reilly Media, 2011, xxiii, 433 p. ISBN 978-0-735-64980-4.

[13] KERZNER, Harold. *Project management: a systems approach to planning, scheduling, and controlling*. 8th ed. New York: Wiley, 2003, xx, 891 s. ISBN 04-712-2577-0.

[14] *A guide to the project management body of knowledge: (PMBOK guide)*. 4th ed. Newton Square: Project Management Institute, 2008, xxvi, 467 s. ISBN 978-1-933890-51-7.

[15] VAN DER MOLEN, Fred. *Get ready for cloud computing: a comprehensive guide to virtualization and cloud computing*. 1st ed. Zaltbommel: Van Haren Publishing, 2010. ISBN 978-90-8753-640-4.

[16] JAMSA, Kris. *Cloud computing: SaaS, PaaS, IaaS, virtualization, business models, mobile, security and more*. 1st ed. Burlington, MA: Jones, 2013, xix, 322 p. ISBN 978-1-4496-4739-1.

[17] MELL, Peter a Timothy GRANCE. *The NIST definition of Cloud Computing*. Gaithersburg, MD: National Institute of Standards and Technology, 2011. Dostupné z: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[18] WILDER, Bill. *Cloud architecture patterns*. 1 st ed. Sebastopol, CA: O'Reilly Media, 2012, xviii, 161 pages. ISBN 978-1-449-31977-9. Dostupné z: http://it-ebooks.info/book/947/.

[19] HUGOS, Michael H. a Derek HULITZKY. *Business in the cloud: what every business needs to know about cloud computing* [online]. New York: Wiley, 2011, xviii, 205 p. [cit. 2014-01-04]. ISBN 978-0-470-91702-2. Dostupné z: http://it-ebooks.info/book/2048/.

[20] CHU-CARROLL, Mark C. *Code in the cloud: programming Google App Engine* [online]. Raleigh, NC: Pragmatic Bookshelf, 2011, x, 306 p. [cit. 2014-01-04]. ISBN 978-1-934356-63-0. Dostupné z: http://it-ebooks.info/book/10/.

[21] MCGRATH, Michael P. *Understanding PaaS* [online]. Sebastopol, CA: O'Reilly Media, 2012, 37 p. [cit. 2014-01-04]. ISBN 978-1-4493-2342-4. Dostupné z: http://it-ebooks.info/book/489/.

[22] REESE, George. *Cloud application architectures: building applications and infrastructure in the Cloud*. Sebastopol, CA: O'Reilly Media, Inc., 2009, xii, 189 p. ISBN 978-0-596-15636-7. Dostupné z: http://it-ebooks.info/book/286/.

[23] Defining Data Classes with JDO - Java. *Google Developers* [online]. 2014-05-06 [cit. 2014-05-17]. Dostupné z: http://developers.google.com/appengine/docs/java/datastore/jdo/dataclasses.

[24] Java Datastore API - Java. *Google Developers* [online]. 2014-05-06 [cit. 2014-05-17]. Dostupné z: https://developers.google.com/appengine/docs/java/datastore/.

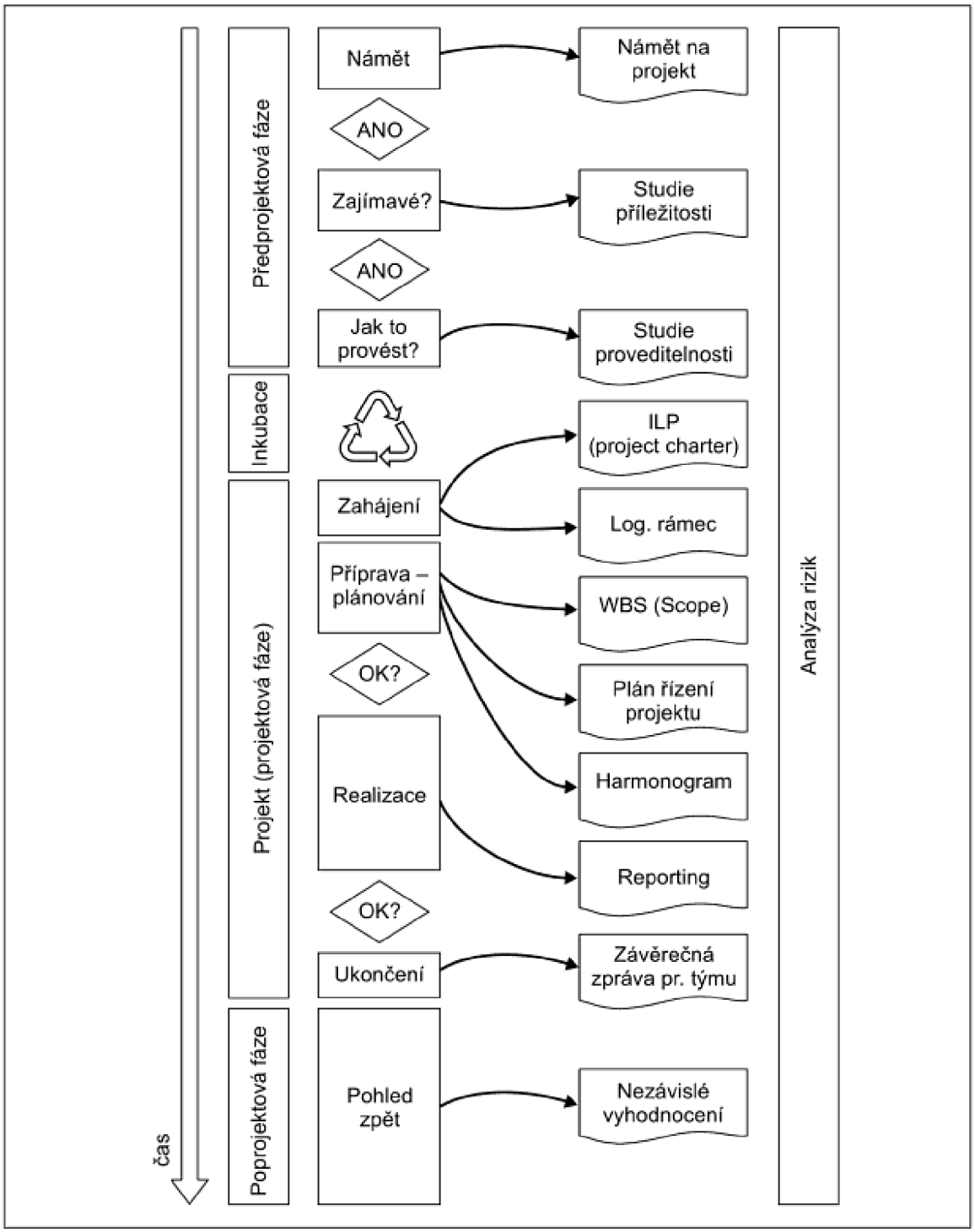
[25] Transactions. *Google Developers* [online]. 2014-05-06 [cit. 2014-05-18]. Dostupné z: https://developers.google.com/appengine/docs/java/datastore/transactions.

[26] Datastore Indexes. *Google Developers* [online]. 2014-05-06 [cit. 2014-05-18]. Dostupné z: https://developers.google.com/appengine/docs/java/datastore/indexes.

[27] Google Cloud Messaging for Android. *Android Developers* [online]. 2014 [cit. 2014-05-19]. Dostupné z: http://developer.android.com/google/gcm/index.html.

Příloha A – Diagram životního cyklu projektu podle IPMA

Následující obrázek je převzat z [10] a zachycuje kompletní model životního cyklu projektu, jak jej definuje IPMA (International Project Management Association).



Příloha B – Obsah CD

Elektronická verze technické zprávy této diplomové práce se nachází v kořenovém adresáři přiloženého média. Je zde ve formátu `pdf` a ve verzi určené pro úpravy ve formátu `docx`. Na tomto médiu se dále nachází následující adresáře:

- `AybeeCloud` – zdrojové soubory projektu z prostředí Eclipse pro klientskou část aplikace
- `AybeeCloud-AppEngine` – zdrojové soubory projektu pro serverovou část aplikace
- `Demo` – adresář s instalačním souborem mobilní aplikace - `AybeeCloud.apk`