

Tvorba systému pro správu hlášení o selhání mobilních aplikací

Crash Reporting System for Mobile Applications

Bc. Pavel Čech

Diplomová práce
2014



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Pavel Čech**
Osobní číslo: **A12456**
Studijní program: **N3902 Inženýrská informatika**
Studijní obor: **Informační technologie**
Forma studia: **kombinovaná**

Téma práce: **Tvorba systému pro správu hlášení o selhání mobilních aplikací**

Zásady pro vypracování:

1. Vypracujte literární rešerši na téma zachytávání pádů aplikací.
2. Popište možnosti zachytávání pádů v systémech iOS, Android a Windows Phone.
3. Analyzujte současný stav řešení zpracování zpráv o pádu mobilních aplikací.
4. Navrhněte informační systém pro obsluhu zpráv o pádu mobilních aplikací na platformě Google App Engine.
5. Součástí bude také ukládání reportu do databáze, vizualizace jejich statistik a odesílání upozornění vývojáři.
6. Zhodnoťte navržené řešení z hlediska bezpečnosti, spolehlivosti a škálovatelnosti.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: tištěná/elektronická

Seznam odborné literatury:

1. **KUMAR, Rob Napier and Mugunth.** IOS 6 Programming Pushing the Limits Advanced Application Development for Apple iPhone, iPad, and iPad Touch. New York: Wiley, 2012. ISBN 978-111-8449-974.
2. **FALAFEL SOFTWARE.** Pro Windows Phone App Development. 3. vyd. New York: Apress, 2013. ISBN 9781430247838.
3. **SATYA KOMATINENI, Dave MacLean a Eric Franchomme TECHNICAL REVIEWERS.** Pro Android 4. New York: Apress, 2012. ISBN 978-143-0239-314.
4. **Programming Google App Engine.** Sebastopol: O'Reilly Media, Inc, 2010. ISBN 978-144-9383-039.
5. **Essential App Engine: Building High-Performance Java Apps with Google App Engine.** Indiana: Addison-Wesley, 2011. ISBN 9780132484756.
6. **SALZ, Peggy Anne a Jennifer MORANZ.** The everything guide to mobile apps: a practical guide to affordable mobile app development for your business. Avon, Mass.: Adams Media, 2013, 303 p. Everything series. ISBN 14-405-5533-8.

Vedoucí diplomové práce:

Ing. Milan Navrátil, Ph.D.
Ústav elektroniky a měření

Datum zadání diplomové práce:

21. února 2014

Termín odevzdání diplomové práce:

20. května 2014

Ve Zlíně dne 21. února 2014

prof. Ing. Vladimír Vašek, CSc.
děkan

L.S.

doc. Mgr. Roman Jasek, Ph.D.
ředitel ústavu

ABSTRAKT

Tato diplomová práce se zabývá tvorbou systému pro zachytávání a zpracování zpráv o pádu mobilní aplikace. V teoretické části analyzuje současný stav řešení a možná dostupná řešení. Rozebírá také platformu Google App Engine, na kterou je práce zaměřena. Dále poskytuje základní informace o mobilních platformách iOS, Android a Windows Phone pro účely zachycení pádu aplikace.

V praktické části je zpracováno zachytávání pádů na třech uvedených platformách a odesílání reportů na vlastní server včetně přidružených informací. Rozebírá také zpracování dat na službě využívající Google App Engine a zobrazení statistických dat o pádech aplikací.

Klíčová slova: mobilní aplikace, Google App Engine, zachytávání pádů

ABSTRACT

This thesis deals with crash reporting system for mobile applications. In the theoretical part it analyzes the current situation and possible solutions available. Thesis also discusses Google App Engine platform on which work is focused. It also provides basic information on mobile platforms as iOS, Android and Windows Phone for catch application crash.

Practical part contains capturing application crashes on three platforms, and sending reports to custom server, including associated information. It also analyzes data processing service using Google App Engine and display statistical data about application crashes.

Keywords: mobile applications, Google App Engine, crash reporting

Poděkování

Děkuji vedoucímu diplomové práce Ing. Milanu Navrátilovi, Ph.D. za ochotu, čas a připomínky k vypracování této práce.

Motto

Není důležité vše vědět, ale mít přehled.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....
podpis diplomanta

OBSAH

OBSAH	7
ÚVOD.....	9
I TEORETICKÁ ČÁST	11
1 PÁDY MOBILNÍCH APLIKACÍ.....	12
1.1 ANDROID.....	12
1.2 iOS.....	13
1.3 WINDOWS PHONE.....	15
2 PROSTŘEDKY PRO SYSTÉM ZACHYTÁVÁNÍ PÁDŮ	16
2.1 VÝJIMKY A CHYBOVÉ SIGNÁLY	16
2.2 TRASOVÁNÍ ZÁSOBNÍKU	17
2.3 INFORMACE O SYSTÉMU A APLIKACI.....	17
2.4 DROBEČKY	18
2.5 SYMBOLIKACE.....	18
2.6 SLUČOVÁNÍ REPORTŮ DO SKUPIN	19
2.7 IDENTIFIKACE UŽIVATELE A ZAŘÍZENÍ.....	19
2.8 NOTIFIKACE VÝVOJÁŘE.....	19
2.9 ZOBRAZENÍ STATISTIK.....	20
2.10 DOPLŇUJÍCÍ POZNÁMKY	20
2.11 INTERAKCE S UŽIVATELEM.....	20
3 DOSTUPNÁ ŘEŠENÍ.....	23
3.1 STANDARDNÍ VÝVOJÁŘSKÉ ROZHRAŇÍ	23
3.2 SOUČASNÝ STAV	25
3.3 ŘEŠENÍ TŘETÍCH STRAN	26
3.3.1 Crittecism	26
3.3.2 Bugsense	27
3.3.3 Crashlytics.....	28
3.3.4 TestFlight	28
3.3.5 HockeyApp	28
3.3.6 QuincyKit.....	29
3.3.7 Flurry.....	29
3.3.8 Google Analytics.....	29
3.3.9 Další řešení třetích stran.....	29
3.4 VLASTNÍ ŘEŠENÍ.....	30
3.4.1 Zachytávání pádů	30
3.4.2 Server	30
4 GOOGLE APP ENGINE.....	31
4.1 PROGRAMOVACÍ JAZYKY.....	31
4.2 ÚLOŽIŠTĚ	31
4.3 OMEZENÍ A LIMITY	32
4.4 BEZPEČNOST	32
4.5 ŠKÁLOVATELNOST	32

4.6	DOSTUPNOST.....	32
II	PRAKTICKÁ ČÁST.....	33
5	MOBILNÍ APLIKACE.....	34
5.1	ANDROID.....	34
5.2	IOS.....	35
5.3	WINDOWS PHONE.....	37
6	ZACHYTÁVÁNÍ PÁDŮ.....	39
6.1	ANDROID.....	39
6.2	IOS.....	39
6.3	WINDOWS PHONE.....	40
7	ODESÍLÁNÍ REPORTŮ Z MOBILNÍ APLIKACE.....	42
7.1	ODESÍLANÉ INFORMACE	42
7.2	ANDROID.....	42
7.3	IOS.....	44
7.4	WINDOWS PHONE.....	45
8	ZPRACOVÁNÍ PÁDU NA SERVERU.....	48
8.1	GOOGLE APP ENGINE	48
8.1.1	Google účet	48
8.1.2	Vytvoření GAE aplikace	49
8.1.3	Google App Engine Launcher	50
8.1.4	Webapp2	50
8.2	NASTAVENÍ APLIKACE.....	50
8.3	OBSLUHA REPORTU	51
8.4	DATASTORE	52
8.5	NOTIFIKACE VÝVOJÁŘE.....	52
8.6	ZOBRAZOVÁNÍ STATISTIK.....	53
8.7	WEBOVÁ SPRÁVA SERVEROVÉ APLIKACE	54
8.7.1	Všeobecný přehled	55
8.7.2	Přehled limitů	55
8.7.3	Správa Datastore	56
9	MOŽNOSTI ROZŠÍŘENÍ.....	57
9.1	PŘIDÁNÍ NOVÝCH PARAMETRŮ	57
	ZÁVĚR	58
	ZÁVĚR V ANGLIČTINĚ.....	60
	SEZNAM POUŽITÉ LITERATURY.....	62
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....	65
	SEZNAM OBRÁZKŮ	66
	SEZNAM TABULEK.....	68
	SEZNAM PŘÍLOH.....	69

ÚVOD

Práce je rozdělena na dvě části. První je teoretická část. Teoretická část popisuje základní informace o pádech mobilních aplikací z pohledu jak uživatele, tak vývojáře. Dále popisuje pojmy a prostředky používané při zachytávání a zpracování pádů aplikací. Teoretická část také popisuje nejpoužívanější aplikace a nástroje třetích stran využívané vývojáři mobilních aplikací. Jsou zde naznačeny způsoby vlastního řešení systému pro zachytávání a zpracování pádů mobilních aplikací. Poslední bod teoretické části popisuje platformu Google App Engine [15], na které je postavena serverová část vlastního systému. Praktická část nejprve popisuje způsoby zpracování pádů na jednotlivých mobilních operačních systémech. Po zpracování pádu následuje část, kde jsou odesílány zachycené reporty na vlastní server. Konečným prvkem systému je zpracování reportů na vlastním serveru. V závěru práce jsou naznačeny možnosti škálovatelnosti vytvořeného systému.

V teoretické části je popis pádů mobilní aplikace rozebrán z pohledu uživatele ve smyslu chování operačního systému. Z pohledu vývojáře práce popisuje činnosti operačního systému na pozadí, tedy to co je skryto běžnému uživateli. Přesto že mobilní operační systémy dokáží zpracovat a odesílat reporty o pádech, je vysvětleno proč nestačí ponechat vše jen na operačním systému. Dále jsou vysvětleny pojmy a prostředky pro systém zpracování pádů mobilních aplikací. To znamená popis toho jakými způsoby a co je vlastně zpracováváno. V dalším celku je popsáno současné řešení a nejvíce používaná řešení třetích stran jako např. Crittecism [8], Bugsense [32] nebo Crashlytics [7]. Kromě řešení, která jsou určena přímo pro zpracování pádů a výjimek v aplikacích práce zmiňuje komplexnější řešení určená pro celkovou analýzu běhu aplikací. Mezi tato řešení patří např. Google Analytics pro mobilní zařízení [17] nebo Flurry [13]. Jeden z hlavních úkolů této práce je vytvoření vlastního řešení systému zachytávání a zpracování pádů. Cílem práce je mimo jiné vytvořit serverové řešení, které bude spolehlivé, škálovatelné a přitom nebude stát mnoho peněz, nejlépe však bude zdarma. Jedna ze zajímavých možností, které se pro tyto účely nabízí je platforma Google App Engine, která je popsána v závěru teoretické části. Tato platforma je popsána jak z hlediska možností jako např. programovací jazyky, úložiště, limitů, tak i z hlediska bezpečnosti, spolehlivosti a škálovatelnosti.

Praktická část řeší způsoby zachycení pádu na jednotlivých mobilních platformách. U některých platform jsou k dispozici ověřené knihovny pro zachycení pádů. Je tak popsáno jakým způsobem je implementovat pro dané platformy. Další část se zabývá odesíláním

reportů o pádu aplikace včetně informací o uživateli, zařízení a aplikaci. Kromě toho jsou popsány způsoby zpracování reportů, když aplikace není připojena k síti Internet. Dále práce řeší zpracování reportů z mobilních zařízení. Je popsán způsob zabezpečení přenosu zprávy, její zpracování, včetně uložení reportů na serverové úložiště. Na serverové části práce řeší také notifikace vývojáři a zobrazování statistik o pádech aplikací. V závěru praktické části jsou popsány možnosti rozšíření vytvořeného systému.

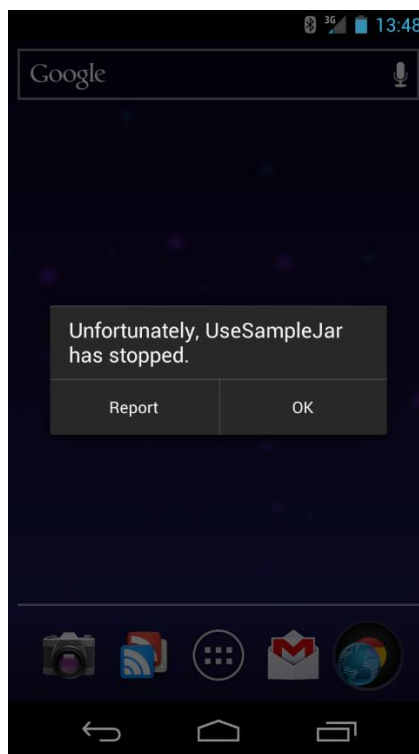
I. TEORETICKÁ ČÁST

1 PÁDY MOBILNÍCH APLIKACÍ

Pádem mobilní aplikace se rozumí její selhání v důsledku chyby aplikace. To má zpravidla za následek okamžité ukončení aplikace operačním systémem. Ukončení aplikace v důsledku pádu může provázet oznamovací dialog například u systému Android, ale také může být aplikace bez jakéhokoli oznámení prostě uzavřena. [24]

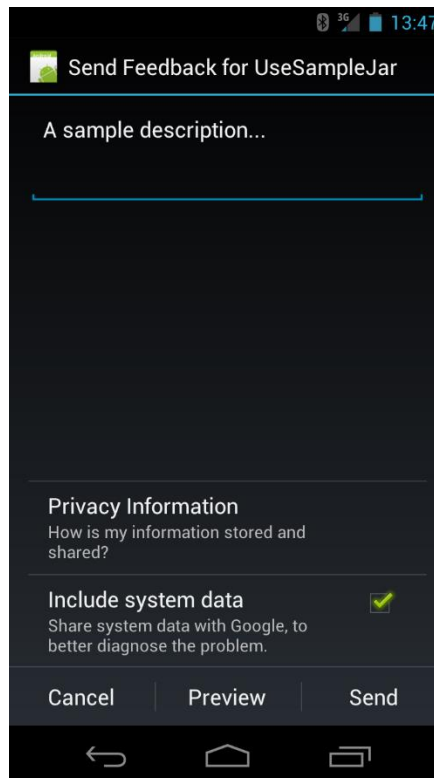
1.1 Android

Pokud nastane pád aplikace u systému Android, je aplikace ukončena a uživateli se zobrazí hláška oznamující, že aplikace neočekávaně skončila. [31]



Obr. 1. Oznamovací dialog o pádu Android aplikace [5]

Jak je vidět na Obr. 1, uživateli se při pádu aplikace zobrazí dialog, který může uživatel odsouhlasit pomocí „OK“, nebo odeslat report pomocí „Report“. Pokud uživatel zvolí „OK“, vývojář již nemá způsob jak report o pádu získat. Pokud však uživatel zvolí „Report“, je přesměrován na další obrazovku (Obr. 2), kde může dopsat vlastní poznámky, může připojit k reportu systémová data a před samotným odesláním si také může zobrazit náhled. [5]



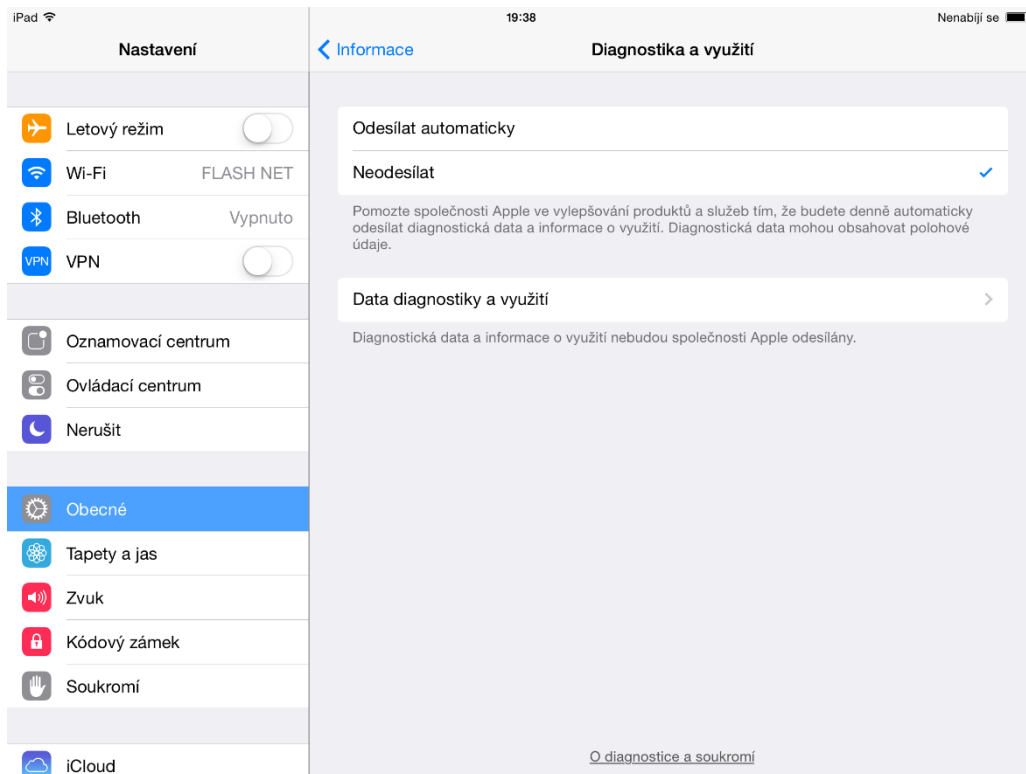
Obr. 2. Android – odeslání reportu [5]

Důležitá informace je, že možnost odeslání reportu je zobrazena jen u těch aplikací, které byly staženy z online obchodu Google Play. Pokud uživatel odešle report, má vývojář možnost vidět tento report na vývojářském rozhraní „Google Play Developer Console“.

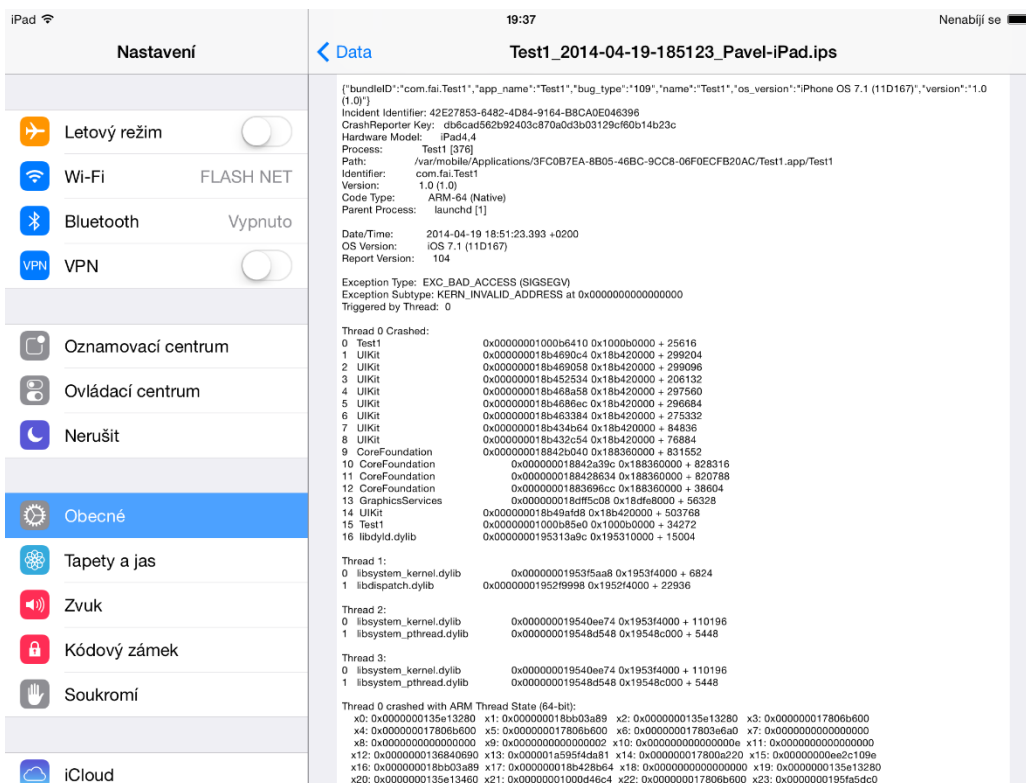
[1, 5]

1.2 iOS

Pokud nastane pád aplikace u systému iOS (verze 5.0 a výše), systém nezobrazí žádnou hlášku, ale jen ukončí aplikaci. Systém iOS však při každém pádu uloží report do zařízení a uživatel si jej pak může dohledat v nastavení telefonu. Reporty lze dohledat konkrétně v „Nastavení → Obecné → Informace → Diagnostika a využití“ pod položkou „Data diagnostiky a využití“. Zde lze také nastavit odesílání diagnostických dat (zahrnuje i reporty) společnosti Apple. Pokud uživatel povolil odesílání diagnostických dat a uživatel následně synchronizuje své zařízení s programem iTunes, Apple pak posílá reporty vývojáři na vývojářské rozhraní iTunes Connect. [24, 2]



Obr. 3. Nastavení odesílání diagnostických dat v systému iOS 7.1



Obr. 4. Detail reportu o pádu v zařízení iOS 7.1

1.3 Windows Phone

Pokud nastane pád u systému Windows Phone (verze 7.0 a výše), systém ukončí aplikaci stejným způsobem jako u iOS, tedy bez dialogu s oznámením o ukončení aplikace. Windows Phone odesílá reporty automaticky, ale za podmínky, že je uživatel připojen k Internetu přes WiFi a povolil zpětnou vazbu společnosti Microsoft v nastavení telefonu. Toto nastavení se nachází konkrétně v „Nastavení → Zpětná vazba“. [23]



Obr. 5. Nastavení zpětné vazby v systému Windows Phone 8.1

2 PROSTŘEDKY PRO SYSTÉM ZACHYTÁVÁNÍ PÁDŮ

Prostředky pro systém zachytávání pádů se rozumí způsoby zachytávání pádů, vytváření a zpracování následných reportů. Reporty o pádech aplikací vytvářené systémem obsahují jen omezené množství informací. Pro vlastní systém však lze využít rozšířených možností, jako jsou např. „drobečky“, identifikace uživatele, doplňující poznámky, nebo na straně serveru pak notifikace vývojáře. [1, 14, 18]

2.1 Výjimky a chybové signály

Základem pro zachycení pádu mobilní aplikace je zachycení neošetřené výjimky a případně chybových signálů. Pro tyto účely mobilní operační systémy poskytují obvykle rozhraní pro obsluhu těchto výjimek a signálů. V případě systému iOS může vypadat obsluha neošetřených výjimek takto: [14, 11, 9]

```
1  NSSetUncaughtExceptionHandler(&HandleException);
2  signal(SIGABRT, SignalHandler);
3  signal(SIGILL, SignalHandler);
4  signal(SIGSEGV, SignalHandler);
5  signal(SIGFPE, SignalHandler);
6  signal(SIGBUS, SignalHandler);
7  signal(SIGPIPE, SignalHandler);
```

U systému Android je způsob obsluhy neošetřených výjimek a chybových signálů obdobný [29]. Systém Windows Phone nabízí pro neošetřené výjimky metodu: [26]

```
1  private void Application_UnhandledException(object sender,
    ApplicationUnhandledExceptionEventArgs e)
```

Selhání navigace v aplikaci lze pak odchytit v metodě:

```
1  private void RootFrame_NavigationFailed(object sender,
    NavigationFailedEventArgs e)
```

Obě metody se zpravidla nacházejí v „App.xaml.cs“. [26]

Následuje tabulka s popisem jednotlivých chybových signálů v systémech iOS a Android: [11]

Tab. 1. Chybové signály operačních systémů

Signál	Popis
SIGILL	Provedení neplatné instrukce (např. skok na neplatnou adresu).
SIGTRAP	Používaný signál pro nástroje ladění.

SIGABRT	Signál inicializovaný samotným procesem pomocí <code>abort()</code> .
SIGFPE	Chyba desetinné tečky, nebo aritmetické operace.
SIGBUS	Chyba sběrnice. Např. Ukazatel na neexistující adresu.
SIGSEGV	Nastane, pokud proces přistupuje k nevalidní paměti.

2.2 Trasování zásobníku

Trasovací zásobník (angl. „Stack trace“) je pro účely zachytávání pádů jednoduše seznam metod, které byly prováděny než aplikace selhala. Tento seznam je základním prvkem při sestavování reportu o pádu aplikace. Vývojář může podle tohoto seznamu často jednoduše dohledat část programového kódu, ve kterém nastala chyba. [32] Zde je ukázka části reportu o pádu Android aplikace, právě s vypsaným zásobníkem:

```

1  STACK_TRACE=java.lang.RuntimeException: This is a crash
2  at com.fai.test1.app.MainActivity$1.onClick(MainActivity.java:27)
3  at android.view.View.performClick(View.java:4445)
4  at android.view.View$performClick.run(View.java:18429)
5  at android.os.Handler.handleCallback(Handler.java:733)
6  at android.os.Handler.dispatchMessage(Handler.java:95)
7  at android.os.Looper.loop(Looper.java:136)
8  at android.app.ActivityThread.main(ActivityThread.java:5114)
9  at java.lang.reflect.Method.invokeNative(Native Method)
10 at java.lang.reflect.Method.invoke(Method.java:515)
11 at
    com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteIn
    it.java:781)
12 at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:597)
    at dalvik.system.NativeStart.main(Native Method)

```

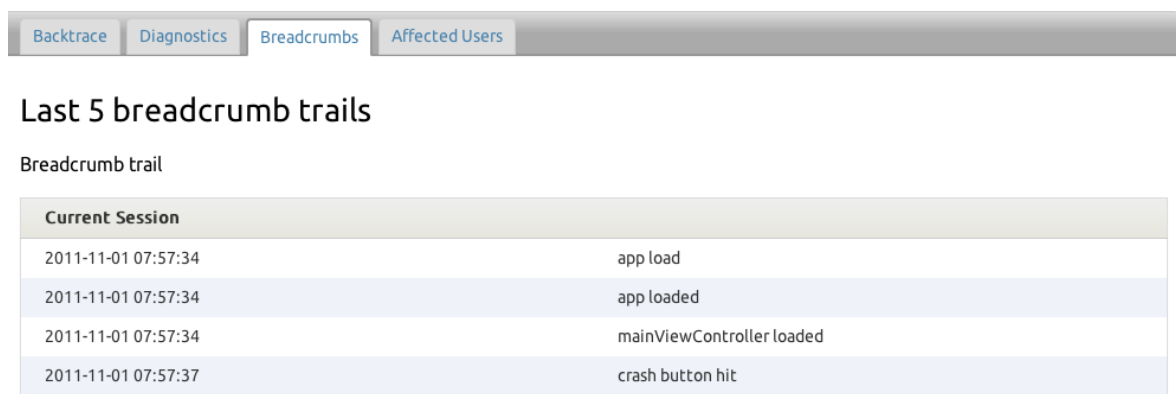
Podle výše vypsaného zásobníku, lze vyvodit, že poslední volaná metoda před pádem aplikace byla „onClick“ v aktivitě „MainActivity“.

2.3 Informace o systému a aplikaci

K tomu, aby byl vývojář mohl identifikovat problém v aplikaci, potřebuje znát verzi aplikace, ve které došlo k pádu a také verzi operačního systému. Součástí reportů mohou být i informace o využití systémových prostředků, jako například využití cpu, paměti a také informace ze sensorů, které jsou v zařízení k dispozici a ke kterým má aplikace oprávněn přístup. [1, 13]

2.4 Drobečky

Drobečky (angl. „Breadcrumb“, resp. „Breadcrumb trails“) jsou doplňkové informace, které se připojují k reportu o pádu. Vývojář tímto způsobem může průběžně v kódu zapisovat stavové informace a v případě pádu aplikace se pak tyto informace připojí k reportu o pádu. To může posloužit vývojáři lepší identifikaci problému v aplikaci, zvláště, když ostatní informace, jako výpis trasovacího zásobníku, nejsou dostatečně nevypovídající o problému způsobující pád. [8]



Obr. 6. Zobrazení drobečků ve webovém rozhraní Crittecism [8]

2.5 Symbolikace

Symbolikace (angl. „Symbolication“) je proces vztahující se k výpisu trasovacího zásobníku. Tento proces je typický zejména pro reporty operačního systému iOS. Pokud vývojář obdrží report o pádu aplikace s výpisem trasovacího zásobníku, získá tento výpis pouze ve formě seznamu hexadecimálních adres. Vývojář však potřebuje místo symbolických adres znát názvy volaných metod v aplikaci. Toho lze dosáhnout procesem zvaným „symbolikace“. Pro tento proces vývojář potřebuje zkompilevanou aplikaci, která byla vystavena na online obchod „App Store“ a soubor *.dSym obsahující data pro proces symbolikace. Nejjednodušší způsob, jak uskutečnit tento proces, je pro iOS vývojáře otevřít vývojové prostředí Xcode, dále otevřít „Window → Organizer → Devices → Device Logs“. Zde lze importovat report a poté provést symbolikaci pomocí „Re-Symbolicate“. [2] Pro ukázkou je uvedena část výpisu trasovacího zásobníku před symbolikací:

```

1 Thread 0 Crashed:
2 Test1 0x000000010004e410 0x100048000 + 25616
3 UIKit 0x000000018d9210c8 0x18d8d8000 + 299208
4 UIKit 0x000000018d92105c 0x18d8d8000 + 299100
5 UIKit 0x000000018d90a538 0x18d8d8000 + 206136
6 UIKit 0x000000018d920a5c 0x18d8d8000 + 297564

```

Po procesu symbolikace vypadá vybraná část výpisu takto:

```
1 Thread 0:
2 Test1 0x000000010004e410 -[ViewController crashTest1_Pushed:]
  (ViewController.m:31)
3 UIKit 0x000000018d9210c4 -[UIApplication
  sendAction:to:from:forEvent:] + 96
4 UIKit 0x000000018d921058 -[UIApplication
  sendAction:toTarget:fromSender:forEvent:] + 20
5 UIKit 0x000000018d90a534 -[UIControl
  _sendActionsForEvents:withEvent:] + 372
6 UIKit 0x000000018d920a58 -[UIControl touchesEnded:withEvent:] +
  580
```

Některé nástroje třetích stran umožňují provést proces symbolikace při zpracování reportu na serveru. To je možné díky tomu, že vývojář při každém vystavení nové verze aplikace na App Store nahraje nový *.dSym soubor na server obsluhující reporty o pádu aplikace. [8]

2.6 Slučování reportů do skupin

Mnohé systémy pro obsluhu reportů pádů aplikací umožňuje třídít reporty do skupin podle typu chybové zprávy. To je užitečné zejména proto, že vývojář se může zaměřit na konkrétní typ problému, který způsobuje pád aplikace. Navíc tak vývojář již nemusí procházet více reportů se stejným typem chyby či problému. [7, 8, 22]

2.7 Identifikace uživatele a zařízení

Součástí reportu jsou zpravidla informace o zařízení. Mezi tyto informace patří například výrobce a model zařízení. Pokud aplikace využívá nějaký unikátní identifikátor zařízení, nebo uživatele, obvykle je tento identifikátor součástí reportu o pádu. Díky jednoznačné identifikaci se pak může vývojář při procházení reportů zaměřit na konkrétní zařízení, nebo konkrétního uživatele. [8]

2.8 Notifikace vývojáře

Aby se vývojář dozvěděl o nových reportech co nejdříve, musí být vhodně notifikován serverem, který zpracovává reporty o pádu. O novém reportu je obvykle vývojář upozorněn pomocí e-mailu. Pokud vývojář odebírá emaily v reálném čase (např. přes push notifikace), může se dozvědět o novém reportu během několika sekund od jeho odeslání ze zařízení, na kterém aplikace selhala. [1]

2.9 Zobrazení statistik

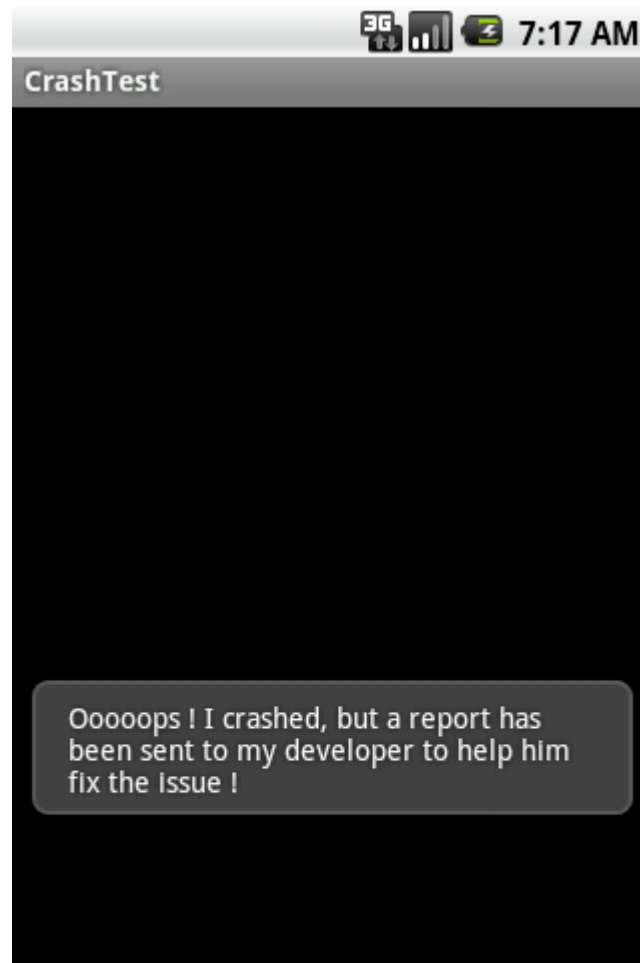
Poté, co jsou reporty zpracovány na serveru a vhodně uloženy do serverového úložiště, je možné na základě těchto dat zobrazovat užitečné statistiky. Součástí takových statistik obvykle bývá tabulkové zobrazení seznamu posledních reportů, včetně informací kolik nastalo pádů za poslední období. [22]

2.10 Doplnující poznámky

Systémy pro obsluhu reportů o pádu aplikace třetích stran obvykle obsahují položku pro přidání vlastní poznámky nebo tagy k reportu. To umožňuje vývojáři lépe identifikovat specifické problémy, způsobujících pád aplikace. [8, 3]

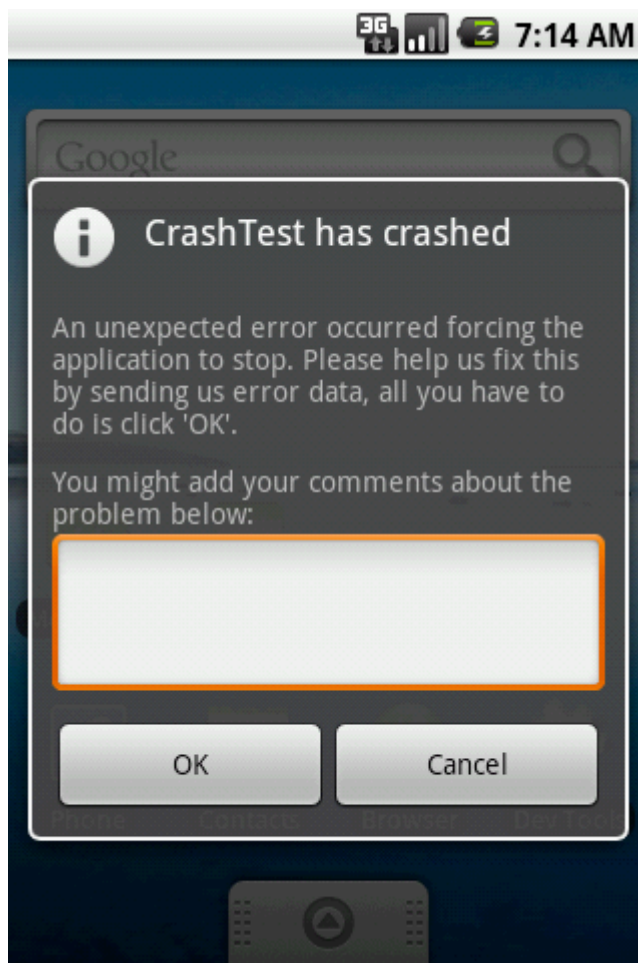
2.11 Interakce s uživatelem

Některé systémy pro obsluhu reportů o pádu aplikace umožňují vybrat si mezi tzv. tichým odesláním a zobrazením dialogu. Při zvolení tichého režimu, probíhá odesílání na pozadí, tedy bez interakce uživatele a také bez jakéhokoliv zobrazení informace o této skutečnosti uživateli. Další možností je odeslání reportu bez potvrzení uživatelem, ale se zobrazením informační zprávy (Obr. 7) [1]



Obr. 7. Zobrazení informace o odeslání reportu uživateli [1]

Kromě toho je také možné uživateli zobrazit dialogové okno s možností potvrzení odeslání reportu, nebo i s možností připsání poznámky přímo od uživatele. Poznámka přímo od uživatele může být velice užitečná, zvláště v případě report jako takový není dost vypovídající o příčině problému. Pokud má však uživatel na výběr mezi odesláním, či neodesláním reportu, zvolí zpravidla druhou možnost. Proto je tento způsob interakce vhodný spíše v případě beta testování aplikace, ještě před vystavením aplikace na online obchod s mobilními aplikacemi. [1]



Obr. 8. Dialog s možností vložení poznámky
uživatele k reportu [1]

3 DOSTUPNÁ ŘEŠENÍ

Ve všech třech mobilních operačních systémech, rozebíraných v této práci, lze využít možností, která již nabízí jednotlivá vývojářská webová prostředí. Výhodou tohoto řešení je, že je funkční ihned po nahrání aplikace do online mobilního obchodu. Toto řešení má však značná omezení, které budou dále rozebrána. Další možností je využít některou z mnohých řešení třetích stran. Existují osvědčená, funkční a velice dobře zpracovaná řešení, z nichž mnohé z nich mají i verzi zdarma se slušnými omezeními. Pokud se ale někdo rozhodne pro vlastní řešení, existují velice užitečné knihovny, které lze pro vlastní systém využít. [1, 24, 25]

3.1 Standardní vývojářské rozhraní

Z úvodního popisu jsou zřejmé některé zásadní problémy řešení na úrovni vývojářských rozhraní. U systému android je po pádu aplikace zobrazen dialog s možností výběru odeslání reportu. Pokud by uživatel vybral kladnou volbu, tedy odeslat report, zobrazí se mu další obrazovka s dalšími volbami. Většina uživatelů neodsouhlasí již první dialog. Ti, kteří odsouhlasí první dialog, z většiny skončí na druhém dialogu. Uživatelé zkrátka nechtějí odesílat, co nemusejí, zvláště pokud jde o osobní údaje. [1] U systémů iOS a Windows Phone musí uživatel odsouhlasit odesílání diagnostických údajů v nastavení telefonu. Toto nastavení však naprostá většina uživatelů vypne. Pokud má uživatel povoleno odesílání diagnostických dat a report ze zařízení uživatele je odeslán, nemá navíc vývojář zaručeno, že se report objeví v jeho vývojářském rozhraní. [23, 24] Konkrétně lze reporty prohlížet pro Android aplikace na Google Play Developer Console, pro iOS aplikace na iTunes Connect a pro Windows Phone aplikace na Windows Phone Dev Center.

Toto řešení má ještě nejméně jednu nevýhodu. Obsah reportu nemůže vývojář nijak ovlivnit. K reportům tak nelze přidat vlastní poznámky, či jednoznačný identifikátor zařízení. [2, 23]

The screenshot shows the Google Play Developer Console interface for the application 'GNUCASH' (org.gnucash.android). The left sidebar contains navigation options: Statistics, Ratings & Reviews, **Crashes & ANRs**, APK, Store Listing, Pricing and Distribution, In-app Products, and Services & APIs. The main content area is titled 'GNUCASH' and 'CRASHES & ANRS'. It includes filters for Type (Crashes, ANRs), Show hidden (YES, NO), Last reported (Last 7 days), and Device. Below the filters, it indicates '1 new crashes' and '3 total crashes'. A table lists the crash reports:

NAME	NEW	REPORTS THIS WEEK	REPORTS TOTAL	LAST REPORTED	HIDE
java.lang.NullPointerException in org.gnucash.android.ui.accounts.AddAccountFragment.loadP...	NEW	4	4	Feb 1, 2013	Hide
java.lang.NullPointerException in android.widget.RemoteViews\$SetOnClickPendingIntent.writeT...		2	3	Feb 1, 2013	Hide
java.lang.NullPointerException in org.gnucash.android.ui.transactions.NewTransactionFragment...		1	9	Jan 29, 2013	Hide

Obr. 9. Procházení reportů na Google play Developer Connsole (zdroj: <http://www.codinguser.com/2013/02/dear-user-please-push-the-report-button/>)

The screenshot displays the iTunes Connect interface for 'ShuffleFrenzy 1.1.2 Crash Reports (Nov 08, 2012)'. It features tabs for different iOS versions: iOS 6.0, **iOS 5.1**, and iOS 5.0. The main section is titled 'Crashes and Freezes' and provides a summary of the most frequent crashes. Below this, there are sections for 'Timeouts' and 'Memory'.

Most Frequent Crashes

Rank	Crash Description	Percentage of Submitted Crashes	Action
1.	libsystem_kernel.dylib: kevent + 24	75%	Download Report
2.	ShuffleFrenzy: 0x38c8	13%	Download Report
3.	ShuffleFrenzy: 0xc3c6	13%	Download Report

Timeouts

Timeout Type	Percentage of Submitted Timeouts	Action
Timeout on Launch	100%	Download Report
Timeout on Quit	0%	No Report Available
Force Quit by User	0%	No Report Available

Memory

These are the memory issues for all versions of your applications.

Resident Private Memory Size

Metric	Value
Average	0.0 B
Maximum	0.0 B

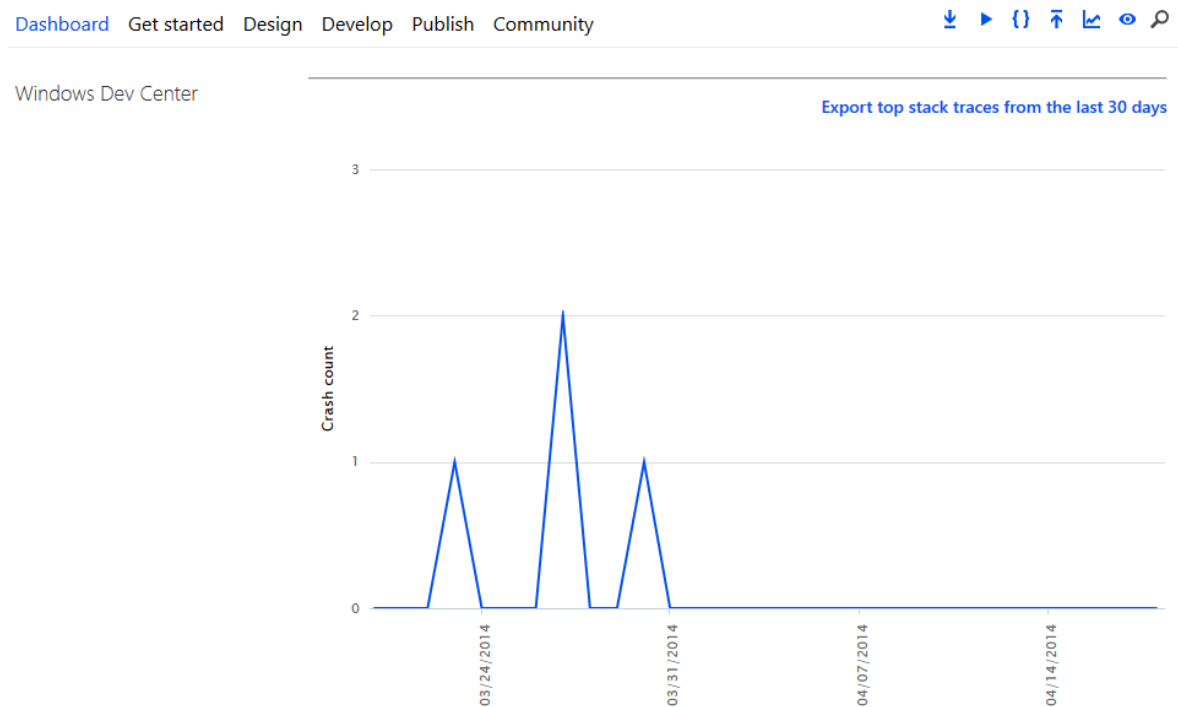
Crashes vs. Memory

This is the ratio of crashes and timeouts to memory issues for all versions of your application.

A horizontal bar chart shows the distribution: 47% Crashes (blue), 53% Timeouts (yellow), and 0% Exhausted Memory (green).

For more information about crash reports, see [Technical Note TN2151](#).

Obr. 10. Procházení reportů na iTunes Connect [9]



Obr. 11. Rozhraní Windows Phone Dev Center pro statistiky reportů

3.2 Současný stav

Cílem této práce je nahradit nevyhovující současná řešení, která jsou navíc řešena jen pro systémy Android a iOS. U některých aplikací není nasazen žádný systém pro reporting, respektive spoléhá na reporty, které jsou vidět ve vývojářském rozhraní. Další právě používané řešení spočívá v zachycení pádu na systémech Android a iOS, odeslání základního reportu bez přidružených informací na vlastní server, který je poté uložen do databáze. Posledním používaným řešením je zachytávání pádů na systémech Android a iOS, odeslání základního reportu bez přidružených informací na vlastní server, který přepošle report vývojářům prostřednictvím e-mailu.

Tato práce si klade za cíl vytvořit vlastní systém pro obsluhu a zpracování reportů na základě znalosti funkčnosti osvědčených řešení třetích stran. Tento systém pak bude implementován pro některé aplikace a po určitém čase bude možné zhodnotit, jestli je takové řešení vyhovující. Pokud vyhovující nebude, pak tato práce pomůže k nalezení lepšího řešení, případně výběru některého řešení od třetí strany.

3.3 Řešení třetích stran

Níže jsou uvedena nejvíce používaná řešení třetích stran. U každého řešení jsou sepsány jeho přednosti, možnosti a také to, jestli dané řešení nabízí některou variantu zdarma. Pokud ano jsou vypsána omezení a limity, které jsou tím podmíněny.

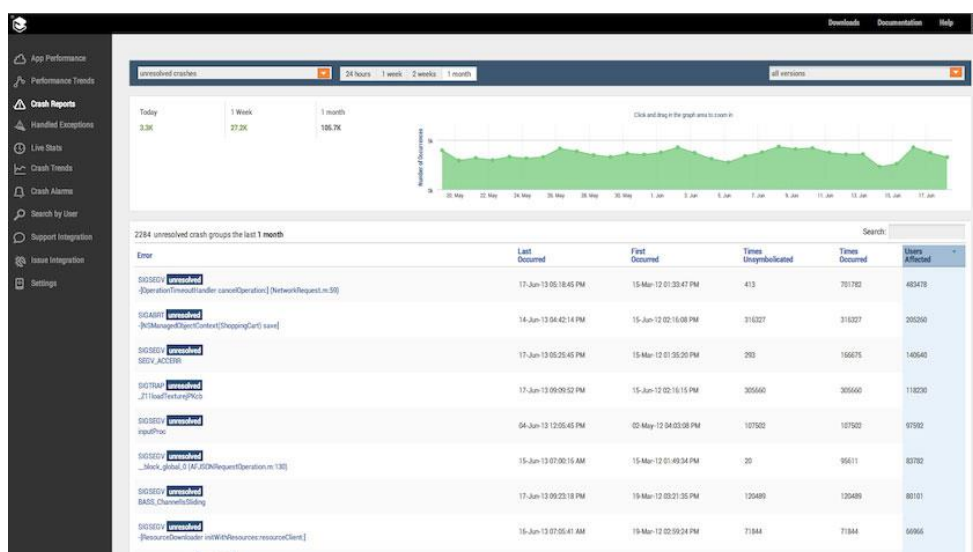
3.3.1 Crittecism

Crittecism je určen pro platformy Android, iOS, Windows Phone a HTML5. [8]

Podporuje monitoring stavu pádů v reálném čase na webu. Kromě obsluhy reportů o pádu dokáže analyzovat síťové operace jednotlivých zařízení a zpracovávat obslužené výjimky. Lze nastavit upozornění při nastavitelném počtu chyb, reportů, výjimek nebo např. i latence síťových operací. Uživatele se stejným problémem rozděluje do skupin, podle typu problému. Umí řadit získaná data podle regionu (získává polohu uživatele), verze operačního systému. Zobrazuje diagnostické údaje využití systémových prostředků. Data o výkonnosti a využití systémových prostředků řadí podle typu zařízení, verze systému, nebo operátora. [8]

Crittecism má více ročních tarifů (1200/6000/7200 dolarů). Kromě toho má také základní tarif zdarma s limitem až 30 tisíc aktivních uživatelů měsíčně. V základním tarifu však nefungují např. drobečky, upozornění, monitoring síťových operací nebo geolokace. [8]

Výčet uživatelů Crittecism obsahuje společnosti Pinterest, Netflix, PayPal, eBay, Yahoo nebo Bloomberg. [8]



Obr. 12. Ukázka webového rozhraní Crittecism [8]

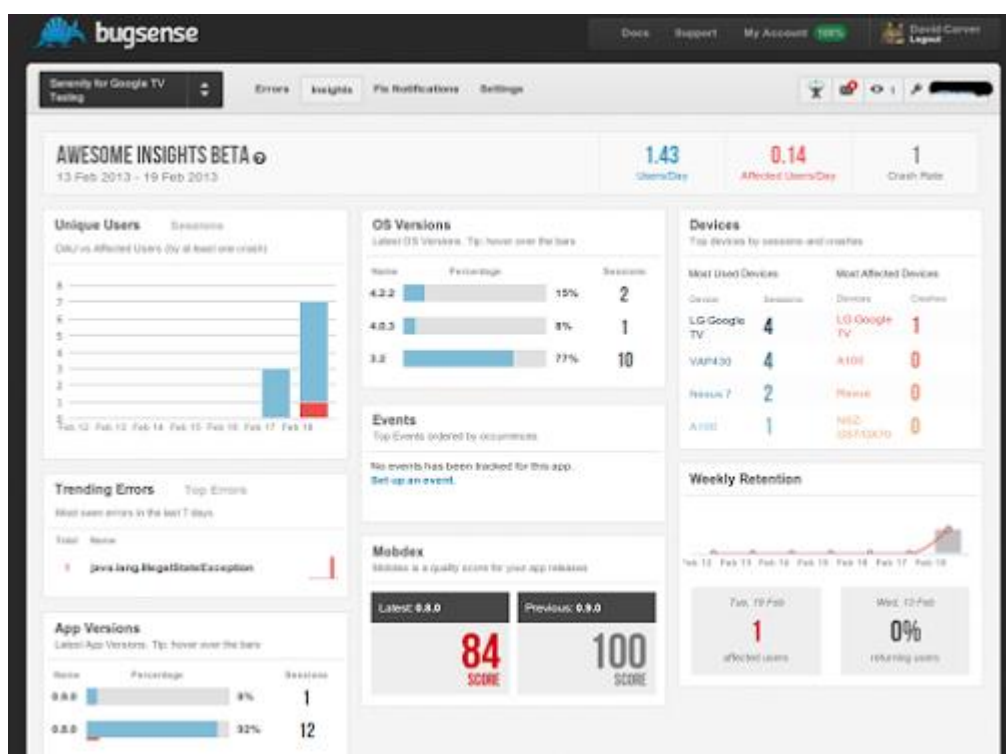
3.3.2 Bugsense

Bugsense je určen pro platformy Android, iOS, Windows Phone, HTML5 a Windows 8. [32]

Využívá jako backend server Google App Engine. U systému Android umožňuje napojit se na knihovnu acra. Na systému iOS dokáže zachytávat logování z NSLog. Podporuje geolokaci zařízení, zobrazení stavu síťové komunikace. Vykresluje graf s počty pádů za zvolené období pro jednotlivé verze aplikace. Umožňuje připojení vlastních poznámek k reportu, zobrazení unikátních uživatelů pro typ chyby, nebo umí zobrazit nejvíce zasažená zařízení. Podporuje nastavení upozornění s možností posílat notifikace jen pro poslední verzi aplikace. [32]

Bugsense má několik měsíčních tarifů (19/99 dolarů a enterprise). Tarif zdarma je omezen na 30 tisíc aktivních uživatelů měsíčně a zároveň až do 500 reportů za měsíc. Verze zdarma také nepodporuje online symbolikaci, vyhledání reportů podle uživatele, drobečky nebo zachycení logování. [32]

Mezi společností využívající Bugsense se řadí například Soundcloud, Microsoft, TechCrunch nebo Mashable. [32]



Obr. 13. Ukázka webového rozhraní Bugsense (zdroj:

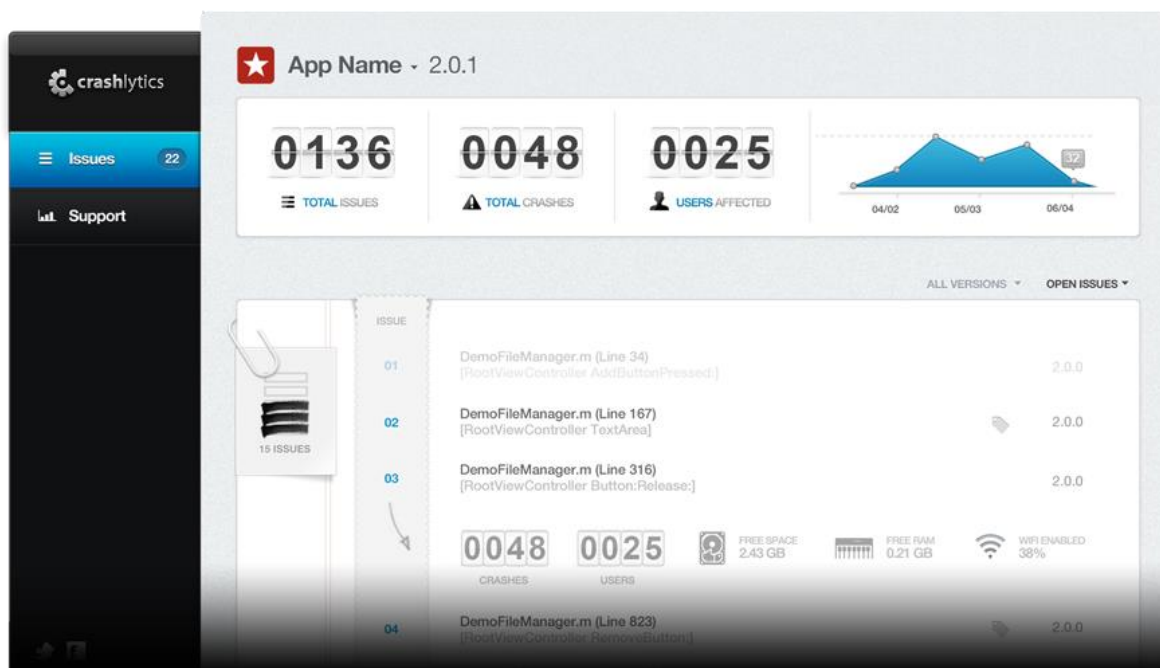
<https://plus.google.com/100427843958963974912/posts/AmjVJA7X7zB>)

3.3.3 Crashlytics

Crashlytics je určen pro platformy iOS a Android. [7]

Podporuje možnost přidání vlastních poznámek k reportu. Spolu s reportem se odesílají informace o typu zařízení, senzoru přiblížení, stav baterie, stav nabíjení nebo orientaci displeje. Součástí reportů je také informace o tom, zda nebyl na zařízení aplikován jailbreak, informace o využití systémových prostředků nebo seznam spuštěných procesů. [7]

Crashlytics je zdarma a využívají ho společnosti jako PayPal, Twitter nebo Kayak. [7]



Obr. 14. Ukázka webového rozhraní Crashlytics [7]

3.3.4 TestFlight

TestFlight je určen pro platformy iOS a Android. Je určen zejména pro beta testování aplikací. Aplikaci lze však distribuovat i do App Store. Zvláštností TestFlight je distribuce OTA, kdy vývojář před samotným beta testováním nahraje na TestFlight, na což jsou notifikováni testéři, že je nová verze aplikace dostupná k otestování. TestFlight je zdarma. [4]

3.3.5 HockeyApp

HockeyApp je určen pro platformy Android, iOS, Windows Phone a Mac. Dělí podobné reporty do skupin. Umožňuje přidání vlastních dat k reportu. Nemá však variantu zdarma. [3]

3.3.6 QuincyKit

QuincyKit je určen pro iOS a OS X. Obsahuje klientský modul pro zachytávání pádů na mobilním zařízení a serverový modul určený pro vlastní PHP server. Zvládá dělení reportů podle typu do skupin, řazení podle verze aplikace nebo zpětnou vazbu uživateli o tom, že problém způsobující pád zařízení byl vyřešen. K zachytávání pádů využívá knihovnu PLCrashReporter. QuincyKit je dostupný zdarma včetně zdrojového kódu. [22]

3.3.7 Flurry

Flurry je určeno pro platformy iOS, Android, Windows Phone, HTML5, BlackBerry a JavaME. Flurry je komplexní analytický nástroj pro mobilní aplikace. K zachytávání pádů na systému iOS využívá knihovnu PLCrashReporter. Služeb Flurry využívají například T-Mobile, Skype, Yahoo, EA, Pepsi, AT&T, eBay nebo AngryBirds. Flurry je zdarma. [13]

First Seen	Last Seen	Crash	First Version Seen	Last Version Seen	Unique Users
12/15/12 0:23.12	12/17/12 4:03.24	NullPointerException	1.2.4	1.2.4	19
12/14/12 4:50.11	12/16/12 16:17.22	NumberFormatException	1.2.4	1.2.4	32
12/13/12 17:25.02	12/16/12 18:21.12	JSONException	1.2.1	1.2.1	19
12/12/12 16:17.12	12/16/12 17:49.40	RuntimeException	1.2.0	1.2.0	12
12/11/12 14:50.31	12/16/12 23:41.10	CastCastException	1.2.4	1.2.4	15
12/10/12 22:03.05	12/16/12 15:32.09	ActivityNotFoundException	1.2.3	1.2.4	31
12/10/12 5:49.22	12/16/12 16:49.54	JSONException	1.2.2	1.2.2	38
12/9/12 12:14.42	12/17/12 0:09.32	NumberFormatException	1.2.3	1.2.3	17
12/8/12 12:44.05	12/17/12 4:02.49	BadAttributeValueExpException	1.2.4	1.2.4	26
12/8/12 1:18.13	12/16/12 22:16.21	IOException	1.2.1	1.2.1	19

Obr. 15. Ukázka přehledu reportů na webovém rozhraní Flurry [13]

3.3.8 Google Analytics

Google Mobile App Analytics je komplexní analytický nástroj obdobně jako Flurry. Poskytuje SDK pro Android a iOS (ve verzi beta). Pro ostatní platformy lze využít odesílání dat přímo pomocí Measurement Protocol. Google Mobile App Analytics je zdarma. [17]

3.3.9 Další řešení třetích stran

Výše byly uvedeny jen ty nejznámější řešení. Existuje však mnoho dalších jako například Jira mobile connect s možností obousměrné komunikace vývojář – uživatel, Apphance používající OTA distribuci, Bug Analytics s odesláním push notifikací o vyřešení bugu, nebo OrionShip. [24, 25]

3.4 Vlastní řešení

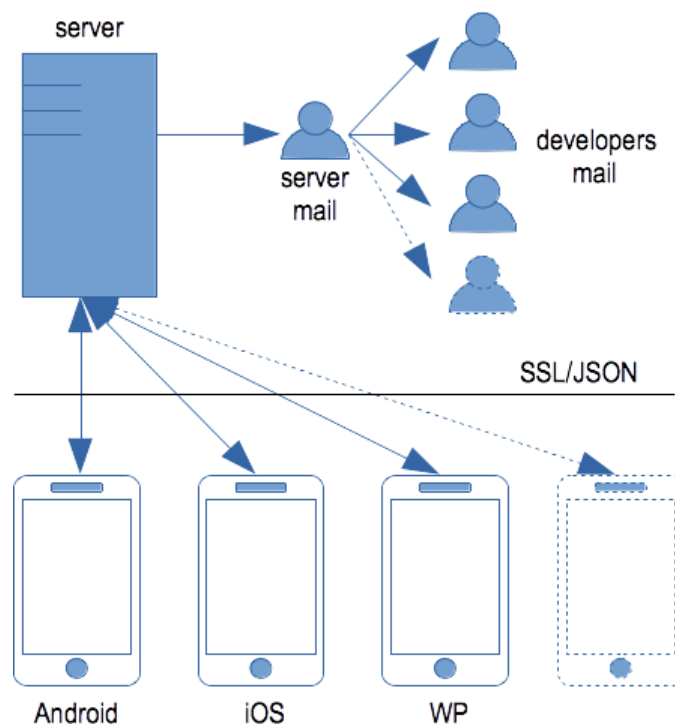
Vlastní řešení se sestává z klientské části, které představují jednotlivé operační systémy a serverové části, která zpracovává reporty odeslané ze všech zařízení.

3.4.1 Zachytávání pádů

Zachytávání pádů je řešeno na systémech Android v jazyce Java, iOS v jazyce Objective-C a Windows Phone v jazyce C# na technologii .NET. Pro účely zachytávání pádů na systému iOS je nevhodnější využít knihovnu PLCrashReporter, kterou využívá také Flurry nebo QuincyKit. U systému Android je použita hojně využívaná knihovna ACRA. [1, 13, 24]

3.4.2 Server

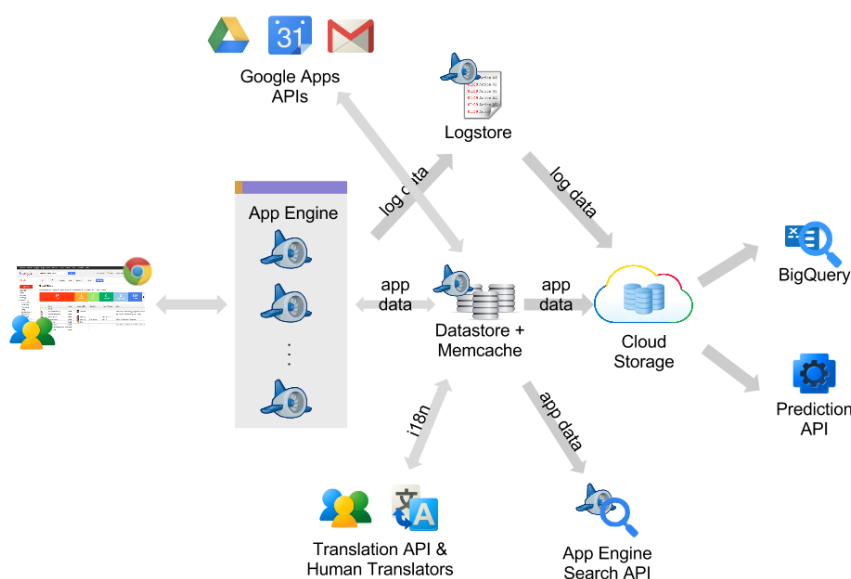
Je možné například využít již hotové řešení od QuincyKit napsané v jazyce PHP s databází MySQL. Pro tuto práci však bude použito platformy Google App Engine, která nabízí zajímavé možnosti hostování vlastní webové služby či aplikace. Tato platforma byla vybrána především kvůli jednoduchosti, bezpečnosti a spolehlivosti služby. Pro komunikaci mezi klientskými aplikacemi a serverem je používán formát JSON a veškerá data jsou přenášena šifrovaným přenosem. [15, 22]



Obr. 16. Diagram systému pro obsluhu pádů
mobilních aplikací

4 GOOGLE APP ENGINE

Google App Engine (dále jen GAE) je platforma umožňující provozovat webové aplikace na Google infrastruktuře. GAE je vhodné pro malé ale i rozsáhlé webové aplikace. GAE vzniklo roku 2008 a nyní je možné říct, že je to ověřená platforma, kterou využívají miliony uživatelů, z toho i mnoho velkých společností. V oblasti systémů pro zpracování reportů o pádech aplikací je to například Bugsense. Kromě toho je GAE platforma, která nabízí propojení s ostatními produkty od Google. GAE aplikace může vytvářet každý, kdo má Google účet. Přitom na jeden Google účet lze vytvořit až 10 aplikací. [15, 32]



Obr. 17. Propojení GAE s ostatním produkty Google [15]

4.1 Programovací jazyky

GAE aplikace je možné psát jazyky Python, Java, PHP nebo Go. Ke každému jazyku je oddělená dokumentace. Nejvíce a nejlépe zdokumentovaný je GAE však pro jazyk Python. Pokud vývojář narazí na problémy, použití jazyka Python, případně Java, bude výhodnější. Důvod je ten, že pro tyto jazyky, zvláště Python, je největší komunita vývojářů, kteří mnoho svých problémů řeší na serveru Stackoverflow a podobných. [15]

4.2 Úložiště

GAE nabízí více způsobů, jak ukládat persistentní data. Pro účely systému pro obsluhu reportů o pádu mobilních aplikací jsou vhodné CloudSQL a Datastore. CloudSQL je zjednodušeně MySQL databáze v cloudu. Datastore je „schema-less NoSQL“ úložiště. Data

do něj nejsou ukládány ve formě tabulek jako u SQL databází, ale jako soubor entit. Datastore je vhodné pro ukládání malého, ale i velice rozsáhlého množství dat. [15]

4.3 Omezení a limity

Ve verzi zdarma má GAE mnohé omezení. Největšími omezeními pro tento projekt jsou početně omezené operace s úložištěm Datastore, omezené množství odesílaných e-mailů, denní příchozí a odchozí datové omezení. Počet operací s úložištěm pro čtení a zápis je shodně 50 000 operací. Úložiště je také omezeno na 1 GB. Pro odesílání e-mailů je omezení 100 e-mailů denně. Pro e-maily, které jsou odesílány na adresy, které byly přednastaveny jako administrátorské, je omezení na 5000 e-mailů denně, ale s omezeným velikostí zprávy oproti běžnému e-mailu. [15]

4.4 Bezpečnost

Komunikace mezi klientem a GAE je šifrována pomocí SSL/TLS. Samotná data v GAE jsou šifrována, zrcadlena na více discích na více lokacích. Při ztrátě dat, nebo poškození disku by tak data neměla být ohrožena a mohou být ihned nahrazena. GAE servery jsou také velice dobře zabezpečeny fyzicky. Bezpečnostní opatření kolem data center by se dala s trochou nadsázky přirovnat k zabezpečení jaderné elektrárny. Co se týče certifikátů, Google Cloud služby jsou certifikovány dle ISO 27001, SSAE 16 Type II a ISAE 3402 Type II. Důležitým faktorem u cloud služeb je však také důvěra v poskytovatele takových služeb. Data mohou být sebelépe zabezpečena, ale když poskytovatel cloudu není důvěryhodný, ztrácí tyto zabezpečení svůj smysl. [15]

4.5 Škálovatelnost

GAE aplikaci je možné vytvořit zdarma pod svým Google účtem. Verze zdarma má mnoho omezení, které v průběhu času nemusí někomu stačit. Proto je možné přejít na prémiový účet, zaplatit si jen to co je opravdu potřeba a množství které je aktuálně potřeba. [15]

4.6 Dostupnost

GAE garantuje dostupnost minimálně 99,9 %. [15]

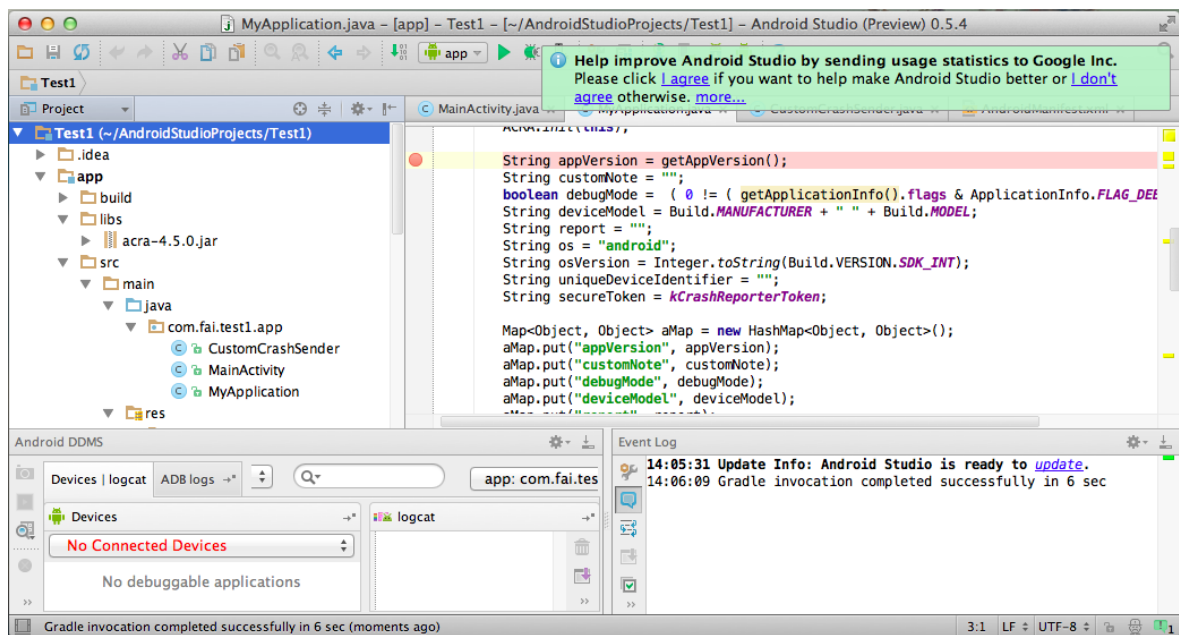
II. PRAKTICKÁ ČÁST

5 MOBILNÍ APLIKACE

Zachytávání pádů je řešeno na systémech Android, iOS a Windows Phone. Pro všechny zmíněné platformy je vytvořena aplikace schopná zachytávat pády a odeslat reporty těchto pádů na vzdálený server. Kromě toho aplikace obsahují jednoduchý kód pro otestování pádu.

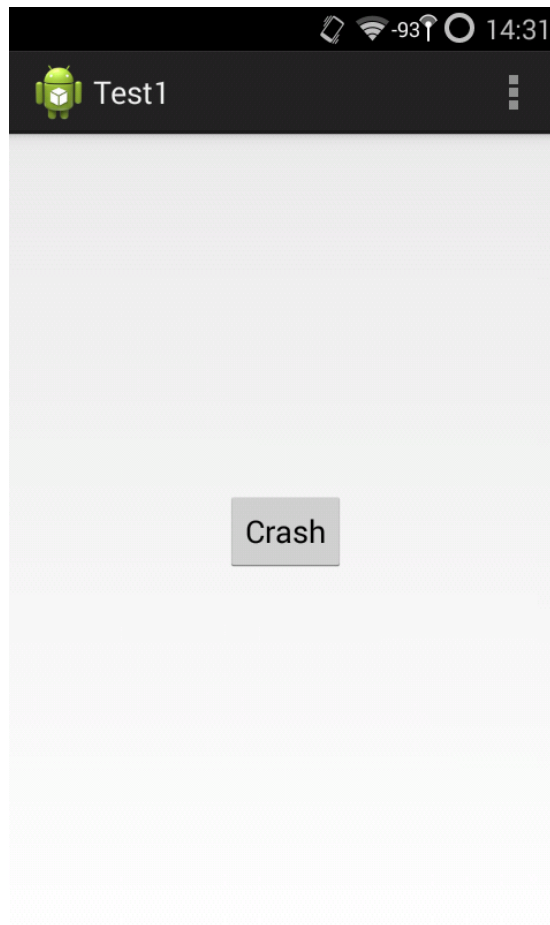
5.1 Android

Pro tvorbu aplikací na systém Android od společnosti Google je doporučeno používat vývojové prostředí Eclipse s rozšířeními pro Android. Stejně tak je ale možné využít nového prostředí Android Studio, které je v současnosti v „preview“ verzi, ale přesto zcela kompletní, funkční vývojové prostředí. Pro vývoj Android aplikace je použit jazyk Java. [15]



Obr. 18. Ukázka použití vývojového prostředí Android Studio

Projekt v Android Studiu 0.5.4 je vytvořen pro systém Android od verze 2.2 s použitím SDK poslední verze, tedy 4.4 KitKat. [15]



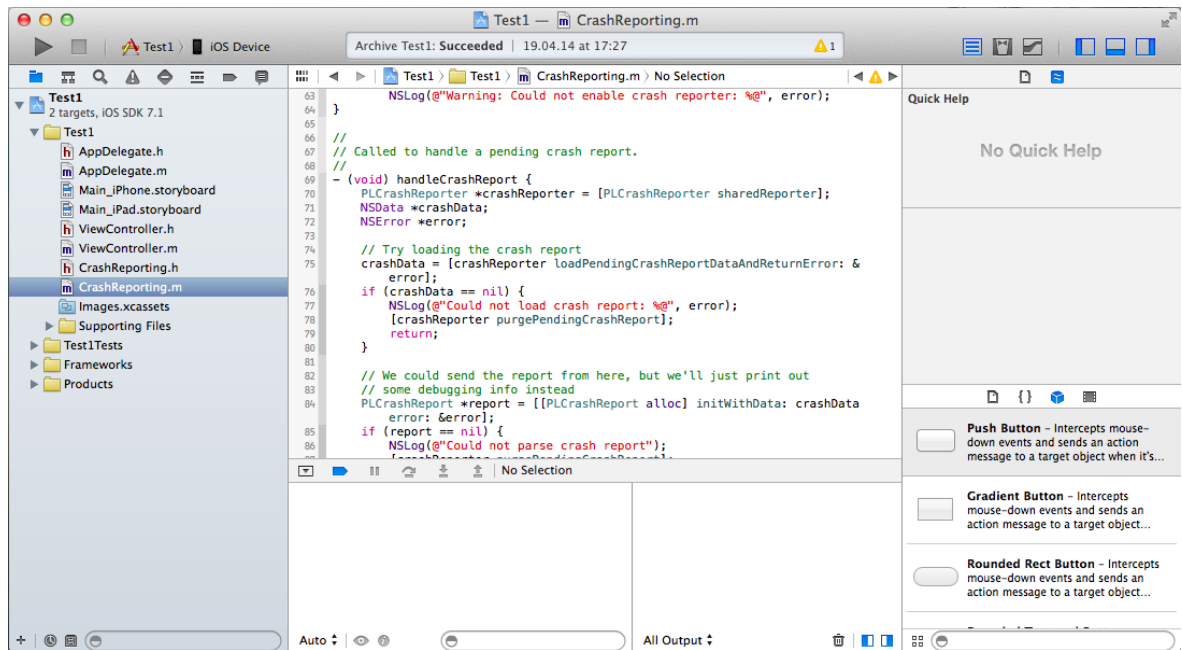
Obr. 19. Testovací Android aplikace

Pro otestování pádu aplikace je v hlavní a zároveň jediné aktivitě aplikace vytvořeno tlačítko s akcí, způsobující pád aplikace:

```
1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3      super.onCreate(savedInstanceState);
4      setContentView(R.layout.activity_main);
5      Button button = (Button) findViewById(R.id.buttonCrash);
6      button.setOnClickListener(new View.OnClickListener()
7      {
8          public void onClick(View v)
9          {
10             throw new RuntimeException("This is a crash");
11         }
12     });
13 }
```

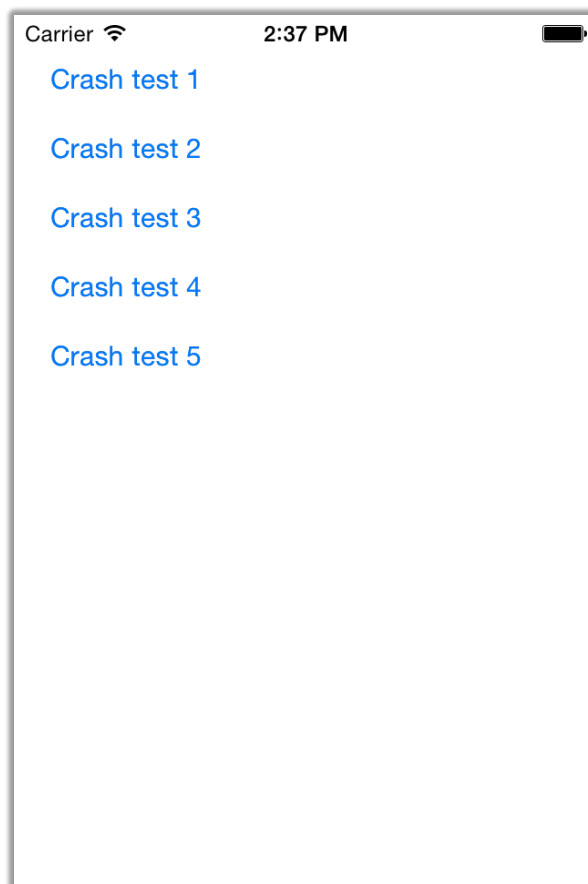
5.2 iOS

Pro tvorbu aplikací na systém iOS od společnosti Apple je určeno vývojové prostředí XCode. Kromě aplikací pro iOS (telefony a tablety), je toto prostředí určeno také pro vývoj desktop aplikací pro systém OS X. Pro vývoj iOS aplikace je použit jazyk Objective-C. [2]



Obr. 20. Ukázka použití vývojového prostředí XCode

Projekt v XCode je vytvořen pro systém iOS (tablet i telefon) od verze iOS 5.1, za použití nejnovějšího SDK verze 7.1. [2]



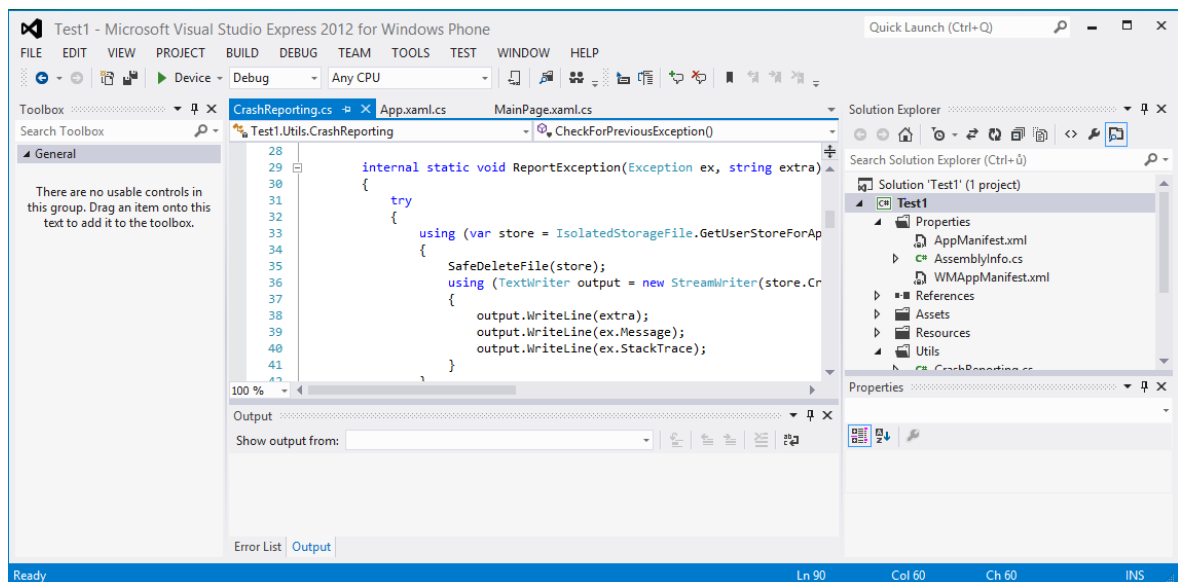
Obr. 21. Testovací iOS aplikace

Pro otestování pádu aplikace je v kódu připraveno několik scénářů způsobujících pád aplikace po stisku jednoho z tlačítek na obrazovce:

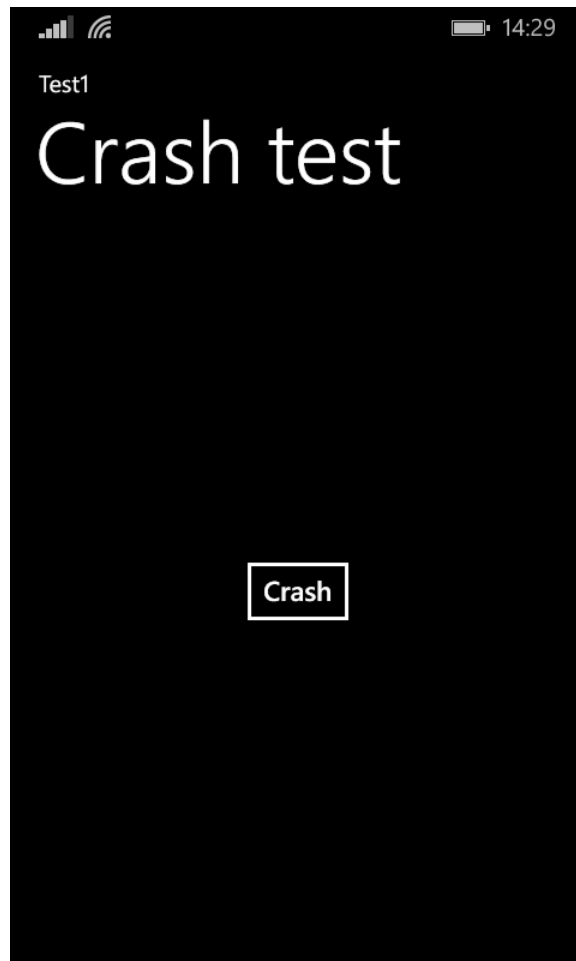
```
1 -(IBAction)crashTest1_Pushed:(id) sender {
2     int *ptr = NULL;
3     printf("%d", *ptr);
4 }
5 -(IBAction)crashTest2_Pushed:(id) sender {
6     UIView *view;
7     [view setFrame:CGRectMake(0,0,0,0)];
8 }
9 -(IBAction)crashTest3_Pushed:(id) sender {
10    int array[100];
11    array[1000] = 23;
12 }
13 -(IBAction)crashTest4_Pushed:(id) sender {
14    [NSException raise:@"Oooops" format:@"%s"];
15 }
16 -(IBAction)crashTest5_Pushed:(id) sender {
17    signal(SIGABRT, nil);
18 }
```

5.3 Windows Phone

Pro tvorbu aplikací na systém Windows Phone od společnosti Microsoft je určeno vývojové prostředí Visual Studio. Pro tuto práci bylo využito konkrétně verze Visual Studio for Windows Phone 2012, které je dostupné zdarma. Pro vývoj aplikace Windows Phone je použit jazyk C#. [23]



Obr. 22. Ukázka použití vývojového prostředí Visual Studio for Windows Phone 2012
Projekt aplikace je vytvořen pro systém Windows Phone verze 8.0 se SDK ve stejné verzi.



Obr. 23 Testovací Windows Phone aplikace

Pro otestování pádu aplikace je v hlavní a zároveň jediné obrazovce aplikace vytvořeno tlačítko s akcí, způsobující pád aplikace:

```
1 private void Button_Click(object sender, RoutedEventArgs e)
2 {
3     throw new FileNotFoundException(@"", new System.Exception());
4 }
```

6 ZACHYTÁVÁNÍ PÁDŮ

Pro zachycení pádů je potřeba odchytit v jednotlivých systémech neošetřené výjimky a chybové signály předcházející pád aplikace. Pro tyto účely jsou pro některé platformy zdarma dostupné knihovny, které jsou schopné tyto výjimky a signály obsloužit. Důležité je také zajistit, aby byly reporty co nejdříve odeslány na sever. Problém nastává, pokud je aplikace off-line. Proto je nutné zachycený report nejdříve uložit do permanentního úložiště mobilního zařízení, pro možnost pozdějšího odeslání. Při vytváření systému pro zachycení výjimek je důležité psát jednoduchý kód s ošetřením všech výjimek, aby aplikace nesehala právě na části kódů, který selhání obsluhuje. [24]

6.1 Android

Pro zachytávání pádů na systému Android je použita knihovna ACRA, která je široce využívána vývojáři na této platformě. ACRA zachytí neobsloužené výjimky a chybové signály a uloží report jako soubor. ACRA již také dokáže odesílat reporty na vlastní server. Pro implementaci knihovny ACRA do vlastního projektu je nutné zdědit třídu „Application“ a nad ní deklarovat cestu k vlastnímu severu: [1]

```
1  @ReportsCrashes (  
2      formUri = "https://pycrashreporter.appspot.com/crash",  
3      reportType = HttpSender.Type.JSON,  
4      httpMethod = HttpSender.Method.POST  
5  )  
6  public class MyApplication extends Application {  
7      @Override  
8      public void onCreate() {  
9          ACRA.init(this);  
10         super.onCreate();  
11     }  
12 }
```

Pokud však chce vývojář odesílat reporty vlastním způsobem s vlastní strukturou odesílaných dat, musí napsat vlastní třídu implementující rozhraní „ReportSender“ a nastavit její instanci pro knihovnu ACRA: [1]

```
1  CustomCrashSender customSender =  
    CustomCrashSender.getInstance(ACRA.getConfig().formUri(), aMap);  
2  ACRA.getErrorReporter().setReportSender(customSender);
```

6.2 iOS

Pro zachytávání pádů na systému iOS je použita knihovna PLCrashReporter. Tuto kvalitní knihovnu používá také například Flurry. Po vlastní implementaci stačí přidat do aplikace dva soubory a přidat jediný řádek kódu do metody: [27]

```

1  (BOOL)application:(UIApplication *)application
   didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
2  {
3      [CrashReporting sharedInstance];
4      return YES;
5  }

```

Inicializace PLCrashReporter se nachází v inicializační metodě třídy CrashReporting:

```

1  (void)initializeReporter {
2  PLCrashReporter *crashReporter = [PLCrashReporter sharedInstance];
3  NSError *error;
4  // Check if we previously crashed
5  if ([crashReporter hasPendingCrashReport])
6      [self handleCrashReport];
7  // Enable the Crash Reporter
8  if (![crashReporter enableCrashReporterAndReturnError: &error])
9      NSLog(@"Warning: Could not enable crash reporter: %@", error);
10 }

```

PLCrashReporter zajišťuje také uložení reportu jako souboru do permanentní paměti. [27]

6.3 Windows Phone

Pro systém Windows Phone je použit vlastní systém zachycení a uložení pádů. Pro implementaci tohoto řešení do aplikace postačuje vložit jediný soubor „CrashReporting.cs“ a do „App.xaml.cs“ přidat do metod: [26]

```

1  private void RootFrame_NavigationFailed(object sender,
   NavigationFailedEventArgs e)

2  private void Application_UnhandledException(object sender,
   ApplicationUnhandledExceptionEventArgs e)

```

tento řádek kódu:

```

1  CrashReporting.ReportException(e.Exception, "");

```

Výše uvedené metody odchytí všechny neobsloužené výjimky. Metoda „ReportException“ poté uloží report jako textový soubor do permanentního úložiště:

```

1  internal static void ReportException(Exception ex, string extra)
2  {
3  try
4  {
5  using (var store =
   IsolatedStorageFile.GetUserStoreForApplication())
6  {
7  using (TextWriter output = new
   StreamWriter(store.CreateFile(filename)))
8  {
9  output.WriteLine(extra);
10 output.WriteLine(ex.Message);
11 output.WriteLine(ex.StackTrace);
12 }

```

```
13 }  
14 }  
15 catch (Exception)  
16 {  
17 }  
18 }
```

[26]

7 ODESÍLÁNÍ REPORTŮ Z MOBILNÍ APLIKACE

Po úspěšném vytvoření a uložení reportu je dalším krokem odeslání reportu na server. V případě, že je aplikace off-line nebo není možné odeslat report ihned po pádu aplikace, je nutné zkontrolovat po každém zapnutí aplikace, jestli není uložen nějaký report v aplikaci a pokusit se jej odeslat. Odesílání probíhá přes zabezpečený přenos HTTPS (TLS/SSL). Report je odesílán v JSON formátu jehož součástí je autentizační token, specifický pro aplikaci.

7.1 Odesílané informace

V JSON objektu jsou odesílána tato data:

Tab. 2. Struktura odesílaných dat

Název	Typ	Popis
appVersion	string	Verze aplikace.
customNote	string	Doplňující poznámka.
debugMode	boolean	Označuje ladící režim.
deviceModel	string	Název modelu zařízení.
report	string	Systémový report pádu.
os	string	Název operačního systému.
osVersion	string	Verze operačního systému
uniqueDeviceIdentifier	string	Unikátní identifikátor.
secureToken	string	Autentizační token.

7.2 Android

Pro odesílání vlastního reportu pomocí knihovny ACRA je potřeba ve třídě „CustomCrashSender“ implementovat metodu „send“:

```

1 public void send(CrashReportData report) throws
  ReportSenderException {
2   try {
3     URL reportUrl;
4     reportUrl = new URL(mFormUri.toString());

```

```
5  JSONObject json = new JSONObject();
6  try {
7    json.put("appVersion", mapJSON.get("appVersion"));
8    json.put("customNote", mapJSON.get("customNote"));
9    json.put("debugMode", mapJSON.get("debugMode"));
10   json.put("deviceModel", mapJSON.get("deviceModel"));
11   json.put("report", report.toString());
12   json.put("os", mapJSON.get("os"));
13   json.put("osVersion", mapJSON.get("osVersion"));
14   json.put("uniqueDeviceIdentifier",
mapJSON.get("uniqueDeviceIdentifier"));
15   json.put("secureToken", mapJSON.get("secureToken"));
16   } catch (JSONException e) {
17   }
18   sendHttpPost(json.toString(), reportUrl);
19 } catch (Exception e) {
20   throw new ReportSenderException("Error while sending report to
Http Post Form.", e);
21 }
22 }
```

Nejprve je tedy vytvořen JSON objekt a ten je poté odeslán pomocí metody „sendHttpPost“:

```
1  private void sendHttpPost(String data, URL url) throws
ReportSenderException {
2    DefaultHttpClient httpClient = new DefaultHttpClient();
3    try {
4      HttpPost httpPost = new HttpPost(url.toString());
5      httpPost.setHeader("Content-type", "application/json;
charset=utf-8");
6      StringEntity se = new StringEntity(data, HTTP.UTF_8);
7      httpPost.setEntity(se);
8      HttpResponse httpResponse = httpClient.execute(httpPost);
9    } catch (ClientProtocolException e) {
10     e.printStackTrace();
11   } catch (UnsupportedEncodingException e) {
12     e.printStackTrace();
13   } catch (IOException e) {
14     throw new ReportSenderException("Error while sending report to
Http Post Form.", e);
15   } finally {
16     httpClient.getConnectionManager().shutdown();
17   }
18 }
```

JSON objekt je tak odeslán v těle POST metody. Důležitá je v tomto procesu výjimka „ReportSenderException“, která rozhoduje o zachování reportu pro další pokus o odeslání, nebo bude smazán. V tomto případě se daná výjimka zavolá, pokud nedojde k úspěšnému spojení se serverem. V případě úspěšného spojení se serverem, ale zároveň s chybovým kódem odpovědi (4xx nebo 5xx), výjimka není zavolána a report se tak nezachová pro pozdější odeslání. Je možné, že report se tak v případě chyby na serveru nikdy neuloží, ale alespoň se zabrání tomu, že by se report odesílal stále znovu, když je na serveru chyba. ACRA volá metodu pro odeslání reportu ihned po pádu aplikace. Pokud se nepodaří odeslat report, zavolá se tato metoda při příštím spuštění aplikace. [1]

7.3 iOS

Ve třídě „CrashReporting“ je po její inicializaci (vytvoření nové instance aplikace) kontrolováno, zda aplikace nemá uložen report z některé z minulých instancí aplikace:

```
1 if ([crashReporter hasPendingCrashReport])
2     [self handleCrashReport];
```

Metoda pro obsluhu reportu „handleCrashReport“ pak vypadá následovně:

```
1 (void) handleCrashReport {
2     PLCrashReporter *crashReporter = [PLCrashReporter sharedReporter];
3     NSData *crashData;
4     NSError *error;
5     crashData = [crashReporter
6         loadPendingCrashReportDataAndReturnError: &error];
7     if (crashData == nil) {
8         [crashReporter purgePendingCrashReport];
9         return;
10    }
11    PLCrashReport *report = [[PLCrashReport alloc] initWithData:
12        crashData error: &error];
13    if (report == nil) {
14        [crashReporter purgePendingCrashReport];
15        return;
16    }
17    [self sendCrashReport:report];
18 }
```

Pokud je systémový report úspěšně otevřen, JSON objekt se sestaví a odešle v metodě „sendCrashReport“:

```
1 (void) sendCrashReport:(PLCrashReport*)report {
2     NSString* crashReport = [PLCrashReportTextFormatter
3         stringValueForCrashReport:report
4         withTextFormat:PLCrashReportTextFormatiOS];
5     NSDictionary *dictData = @{@"appVersion": [[NSBundle mainBundle]
6         infoDictionary] objectForKey:@"CFBundleVersion"},
7         @"customNote": @"",
8         @"debugMode": [NSNumber
9             numberWithBool:kCrashReportIsDebug],
10        @"deviceModel": [self
11            platformString],
12        @"report": crashReport,
13        @"os": @"ios",
14        @"osVersion": [[UIDevice
15            currentDevice] systemVersion],
16        @"uniqueDeviceIdentifier": @"",
17        @"secureToken":
18            kCrashReportWebServiceSecureToken};
19    NSData *jsonData = [NSJSONSerialization dataWithJSONObject:dictData
20        options:0
21        error:nil];
22
23    NSMutableURLRequest *postRequest = [NSMutableURLRequest
24        requestWithURL:[NSURL URLWithString:[NSString
25            stringWithFormat:@"%@", kCrashReportWebServiceURL]]];
```

```

6  [postRequest setValue:@"application/json"
   forHTTPHeaderField:@"Content-Type"];
7  [postRequest setHTTPMethod:@"POST"];
8  [postRequest setHTTPBody:jsonData];
9  self.receivedData = [NSMutableData dataWithCapacity: 0];
10 NSURLConnection *theConnection=[[NSURLConnection alloc]
   initWithRequest:postRequest delegate:self];
11 if (!theConnection) {
12     self.receivedData = nil;
13 }
14 }

```

V případě úspěšného spojení se serverem je zavolána metoda „connectionDidFinishLoading“, kde je odstraněn uložený report pomocí „purgePendingReport“:

```

1  (void)connectionDidFinishLoading:(NSURLConnection *)connection
2  {
3  NSLog(@"Succeeded! Received %lu bytes of data", (unsigned
   long)[self.receivedData length]);
4  NSLog(@"Data: %@", [[NSString alloc] initWithData:self.receivedData
   encoding:NSUTF8StringEncoding]);
5  self.receivedData = nil;

6  PLCrashReporter *crashReporter = [PLCrashReporter sharedReporter];
7  [crashReporter purgePendingCrashReport];
8  }

```

Report se po odeslání a úspěšném spojení smaže v aplikaci i pokud server vrátí chybový kód (4xx, 5xx). Nedbude tak docházet k neustálým pokusům o odeslání reportů, pokud je chyba na serveru. [27]

7.4 Windows Phone

Po novém spuštění aplikace je potřeba zkontrolovat jestli není uložen report o pádu minulé instance aplikace. Proto je do „App.xaml.cs“ zavolána metoda z třídy „CrashReporting“:

```

1  private void Application_Launching(object sender,
   LaunchingEventArgs e)
2  {
3  CrashReporting.CheckForPreviousException();
4  }

```

V metodě „CheckForPreviousException“ je zkontrolováno, jestli existuje soubor s reportem k odeslání:

```

1  string contents = null;
2  using (var store =
   IsolatedStorageFile.GetUserStoreForApplication())
3  {
4  if (store.FileExists(filename))
5  {

```

```
6     using (TextReader reader = new
    StreamReader(store.OpenFile(filename, FileMode.Open,
    FileAccess.Read, FileShare.None)))
7     {
8         contents = reader.ReadToEnd();
9     }
10 }
11 }
```

Pokud soubor se systémovým reportem existuje, sestaví se JSON objekt:

```
1 Newtonsoft.Json.Linq.JObject jobject = new JObject
2 {
3     {"appVersion", appVersion},
4     {"customNote", customNote},
5     {"debugMode", debugMode},
6     {"deviceModel", deviceModel},
7     {"report", report},
8     {"os", os},
9     {"osVersion", osVersion},
10    {"uniqueDeviceIdentifier", uniqueDeviceIdentifier},
11    {"secureToken", secureToken}
12 };
13 string jsonString = jobject.ToString();
```

JSON objekt je poté odeslán na server:

```
1 WebClient webClient = new WebClient();
2 webClient.Encoding = System.Text.Encoding.UTF8;
3 webClient.Headers["Content-Type"] = "application/json; charset=utf-8";
4 webClient.UploadStringCompleted += new
    UploadStringCompletedEventHandler(webClient_UploadCompleted);
5 webClient.UploadStringAsync(new Uri(crashURL), "POST", jsonString);
```

Po odeslání je zkontrolováno, jestli došlo k úspěšnému spojení se serverem a v případě úspěšného spojení je report smazán:

```
1 private static void webClient_UploadCompleted(object sender,
    UploadStringCompletedEventArgs e)
2 {
3     try
4     {
5         if (e.Error != null)
6         {
7             if (e.Error.GetType().Name == "WebException")
8             {
9                 WebException we = (WebException)e.Error;
10                HttpWebResponse response =
    (System.Net.HttpWebResponse)we.Response;
11                if (response.ResponseUri.ToString().Equals(""))
12                    return;
13            }
14        }
15        SafeDeleteFile(IsolatedStorageFile.GetUserStoreForApplication());
16    }
17    catch (Exception)
18    {
19    }
```

20 }

8 ZPRACOVÁNÍ PÁDU NA SERVERU

Jako server byl vybrán GAE. Aplikace napsaná na této platformě se skládá z několika částí. První částí je obsluha reportů s následným uložením reportů do úložiště. Zároveň s tím je odeslána notifikace přes e-mail. Poslední částí je zobrazení statistických dat na základě reportů uložených v úložišti.

8.1 Google App Engine

Pro aplikaci na GAE by použit jazyk Python. Python je nejvhodnější ze všech možností, které GAE nabízí. Část funkcí GAE je popsáno pouze v jazyce Python a v tomto jazyce lze také nalézt nejlepší podporu komunity. [15]

8.1.1 Google účet

Vytvoření GAE aplikace je vázáno na Google účet. Google účet lze vytvořit snadno založením nové e-mailové schránky Gmail. Na jeden Google účet pak lze vytvořit až 10 GAE aplikací. V případě potřeby tak lze velmi snadno vytvořit pro každou novou aplikaci samostatný Google účet a pro každou mobilní platformu jednu GAE instanci aplikace. [15]



Google app engine p3cech@gmail.com | [My Account](#) | [Help](#) | [Sign out](#)

My Applications

« Prev 20 1-1 of 1 Next 20 »

Application	Title	Storage Scheme	Status
pycrashreporter	Crash Reporter	High Replication	Running 

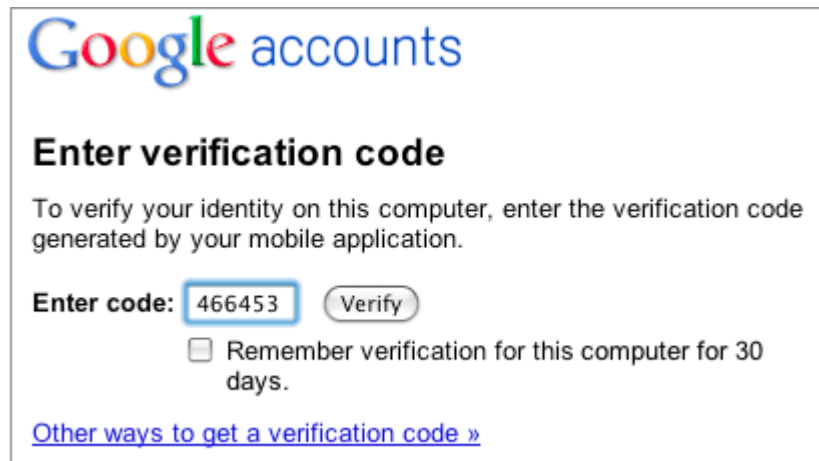
« Prev 20 1-1 of 1 Next 20 »

You have 24 applications remaining.

© 2014 Google | [Terms of Service](#) | [Privacy Policy](#) | [Blog](#) | [Discussion Forums](#) | [Project](#) | [Docs](#)

Obr. 24. Zobrazení seznamu GAE aplikací

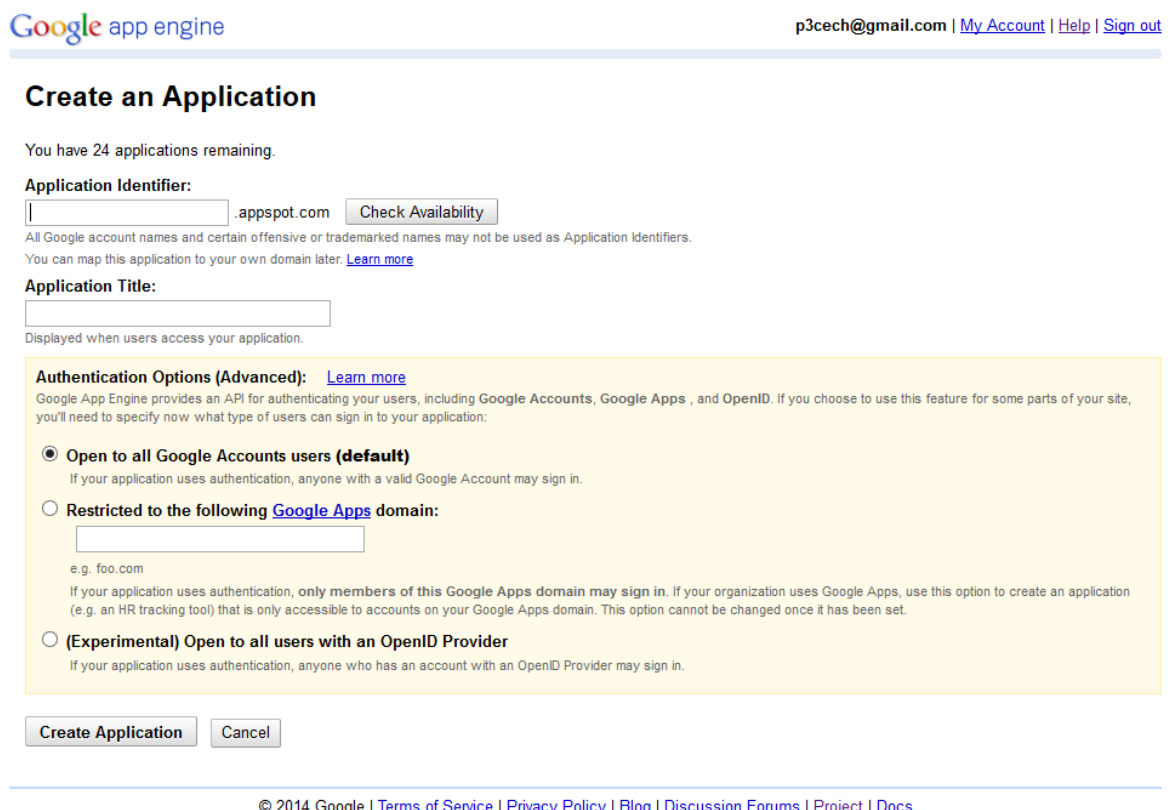
Z důvodu většího zabezpečení přístupu ke GAE aplikacím lze u každého GAE účtu nastavit dvojí ověřování. To znamená standardní ověření pomocí uživatelského jména a hesla a poté pomocí číselného kódu, vygenerovaného a odeslaného na přednastavené telefonní číslo. [15]



Obr. 25. Dvojité ověřování Google účtu (Zdroj: <http://www.lookyourstuff.com/2-step-verification-an-extra-security-layer-for-google-account/1133/>)

8.1.2 Vytvoření GAE aplikace

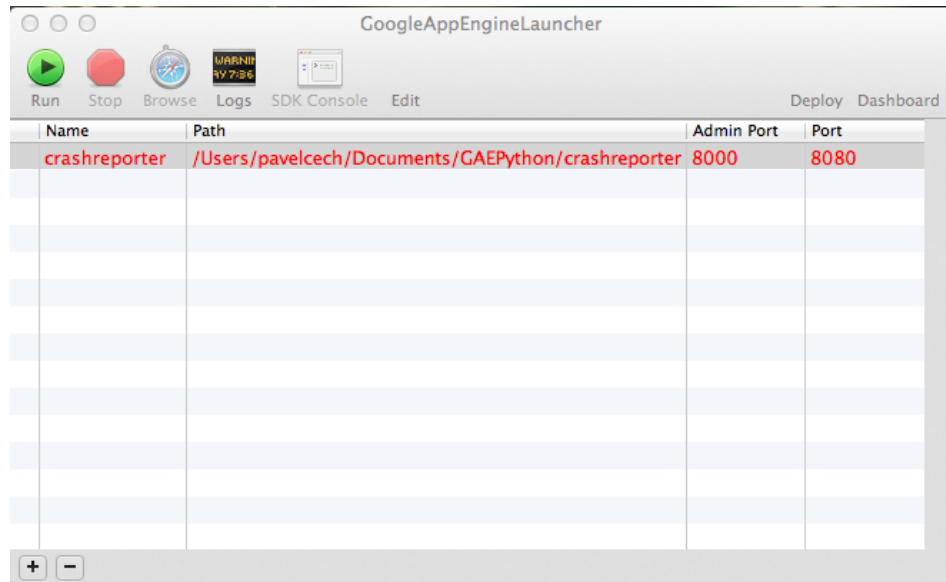
Vytváření nové GAE aplikace spočívá v zadání unikátního názvu domény třetího řádu (subdomény) na doméně „appspot.com“ a názvu aplikace. [15]



Obr. 26. Vytvoření nové GAE aplikace [15]

8.1.3 Google App Engine Launcher

Jako editor pro zdrojový kód napsaný v jazyce Python byl použit textový editor TextWrangler na OS X. Pro testování na lokálním počítači a vystavování aplikace na produkční server byl využit softwarový nástroj Google App Engine Launcher. [15]



Obr. 27. Aplikace Google App Engine Launcher

8.1.4 Webapp2

Pro obsluhu webových požadavků je použita knihovna webapp2. Směrování URL adres je pak definováno následovně:

```

1 application = webapp2.WSGIApplication([
2     ('/', MainPage),
3     ('/crash', HandleCrashReport),
4     ('/detail/(\d+)', CrashDetail)
5     ], debug=True)

```

8.2 Nastavení aplikace

V souboru „App.yaml“ se nachází nastavení aplikace. Jsou zde základní nastavení pro název aplikace, verzi, běhové prostředí Python:

```

1 application: pycrashreporter
2 version: 1
3 runtime: python27
4 api_version: 1
5 threadsafe: true

```

Následují nastavení pro mapování URL adres a statického obsahu:

```
1 handlers:
2 # Static assets
3 - url: /favicon\.ico
4   static_files: favicon.ico
5   upload: favicon\.ico
6 - url: /js
7   static_dir: static/js
8 - url: /css
9   static_dir: static/css
10 - url: /images
11  static_dir: static/images

12 # Endpoints handler
13 - url: /.*
14   script: crashreporter.application
```

Na konci souboru jsou vypsané použité knihovny:

```
1 libraries:
2 - name: webapp2
3   version: latest
4 - name: jinja2
5   version: latest
```

8.3 Obsluha reportu

Pro samotný report je vytvořen model takto:

```
1 class CrashReport(ndb.Model):
2     appVersion = ndb.StringProperty(indexed=True)
3     customNote = ndb.StringProperty(indexed=False)
4     debugMode = ndb.BooleanProperty(default=False)
5     deviceModel = ndb.StringProperty(indexed=False)
6     report = ndb.StringProperty(indexed=False)
7     os = ndb.StringProperty(indexed=True)
8     osVersion = ndb.StringProperty(indexed=True)
9     uniqueDeviceIdentifier = ndb.StringProperty(indexed=True)
10    updateDate = ndb.DateTimeProperty(auto_now_add=True)
```

Nejprve je obsloužena metoda POST pro odeslání reportu, extrahuje se obsah těla zprávy, který je následně deserializován do objektu typu dictionary:

```
1 class HandleCrashReport(webapp2.RequestHandler):
2     def post(self):
3         try:
4             crashReport_dictionary = json.loads(self.request.body)
```

Dříve než je objekt převeden do modelu a uložen do Datastore úložiště, jsou zkontrolovány dva parametry. První je ověřeno, jestli objekt obsahuje správný autentizační token. Pokud je token správný, je ověřeno, zda objekt obsahuje systémový report. Systémový report a autentizační token jsou tak jediné povinné parametry objektu. V případě, že objekt neobsahuje správný autentizační token, neobsahuje systémový report, nebo při deserializaci došlo k chybě, serverová aplikace vrátí chybový kód 400:

```
1 response.set_status(400)
```

V opačném případě aplikace vrátí kód 204.

8.4 Datastore

Původní plán bylo využít pro uložení reportů databázi CloudSQL, ta však na rozdíl od Datastore nemá variantu zdarma.

Pokud byl příchozí report úspěšně deserializován do objektu typu dictionary, převede se objekt do modelu:

```
1 crashReport = CrashReport(parent=default_key())
2 crashReport.report = crashReport_dictionary.get('report')
3 crashReport.appVersion = crashReport_dictionary.get('appVersion')
4 crashReport.customNote = crashReport_dictionary.get('customNote')
5 crashReport.debugMode = crashReport_dictionary.get('debugMode')
6 crashReport.deviceModel = crashReport_dictionary.get('deviceModel')
7 crashReport.os = crashReport_dictionary.get('os')
8 crashReport.osVersion = crashReport_dictionary.get('osVersion')
9 crashReport.uniqueDeviceIdentifier =
  crashReport_dictionary.get('uniqueDeviceIdentifier')
```

Model je zapsán do Datastore úložiště jednoduchým příkazem:

```
1 crashReport.put()
```

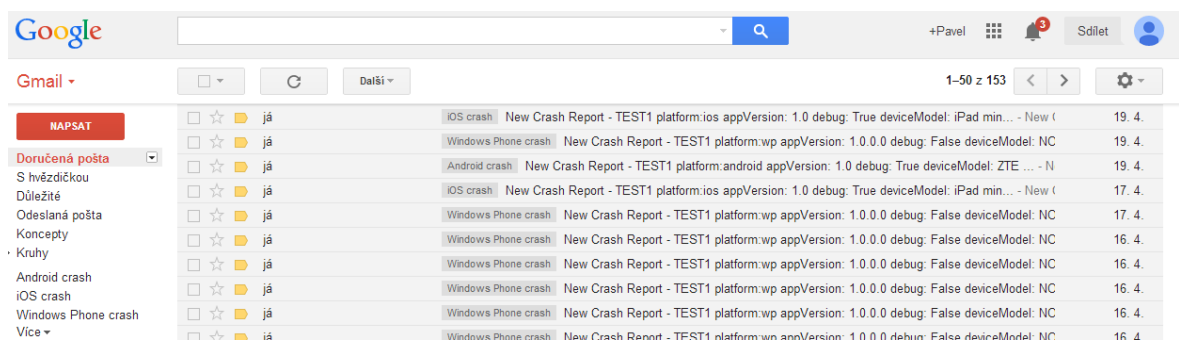
8.5 Notifikace vývojáře

Po úspěšné deserializaci reportu je odeslán e-mail na e-mailovou adresu účtu, pod kterým je spravována GAE aplikace:

```
1 def sendNotificationMail(crashReport, reportString, appURL):
2     try:
3         mail.send_mail(sender="Crash Reporter <p3cech@gmail.com>",
4                         to="Crash Reporter <p3cech@gmail.com>",
5                         subject="New Crash Report - " + str(APP_NAME) + \
6                             " platform:" + str(crashReport.os if crashReport.os
7                             else "unknown") + \
8                             " appVersion: " + str(crashReport.appVersion) + \
9                             " debug: " + str(crashReport.debugMode) + \
10                            " deviceModel: " + str(crashReport.deviceModel) + \
11                            " osVersion: " + str(crashReport.osVersion) + \
12                            " uniqueIdentifier: " +
13                            str(crashReport.uniqueDeviceIdentifier),
14                            body= """"New Crash Report""",
15                            html= """"New Crash Report: <a href="%s">link</a><br
16                                /><pre>%s</pre>"" % (appURL, str(reportString))
17                            )
18     except:
19         logging.error('Cannot send mail with crash report')
```

Na e-mailovém účtu aplikace jsou poté nastaveny filtry, které označují jednotlivé reporty podle názvu mobilního operačního systému v předmětu zprávy. Další filtry poté podle typu systému a aplikace přeposílají e-maily jednotlivým vývojářům.

Počet emailů odeslaných z jedné GAE aplikace je však omezený. V případě běžného e-mailu je tento limit 100 e-mailů denně. V nastavení GAE je možné přidat e-maily administrátorů, kterým lze odeslat denně až 5000 e-mailů. Velikost e-mailů odeslaných administrátorům je však omezená na tolik, že e-mail s delším systémovým reportem se neodešle. [15]



Obr. 28. Označování notificačních emailů s reporty

Pokud má vývojář mobilních aplikací na svém telefonu příjem e-mailů přes protokol Exchange ActiveSync, nebo přijímá e-maily pomocí push notifikací, může se dozvědět o pádu aplikace prakticky okamžitě. To platí zejména pro řešení na systému Android, kde se posílá report na server ihned při pádu aplikace. [15, 2]

8.6 Zobrazování statistik

Před zobrazením statistických dat se musí uživatel přihlásit pomocí Google účtu. Po přihlášení je ověřen účet oproti předdefinovaným administrátorským účtům, případně developerským účtům předdefinovaných v samotné aplikaci:

```

1 user = users.get_current_user()
2 isCurrentUserDeveloper = isUserAppDeveloper(user)
3 if users.is_current_user_admin() or isCurrentUserDeveloper:

```

K získání potřebných údajů je možné využít dotazů volaných oproti úložišti Datastore. Například pro získání počtu přijatých reportů ze systému Android za posledních 48 hodin používá aplikace tento dotaz:

```

1 CrashReport.query(
    ancestor=default_key()).filter(CrashReport.os == 'android',
    CrashReport.updateDate > seconds_ago(48*60*60)).count()

```

Pro získání posledních 200 reportů seřazených podle data uložení je pak použit dotaz:

```

1 reports_query = CrashReport.query(
    ancestor=default_key()).order(-CrashReport.updateDate)
    crashreports = reports_query.fetch(200)

```

Pro zobrazení statistik je využita knihovna pro HTML šablony jinja2. Vypsání HTML tabulky s reporty pomocí jinja2 vypadá takto: [15]

```

1 <table id="reports">
2 <tr>
3   <th>App version</th>
4   <th>Custom note</th>
5   <th>Debug mode</th>
6   <th>Device model</th>
7   <th>Report</th>
8   <th>OS</th>
9   <th>OS version</th>
10  <th>Unique Device Identifier</th>
11  <th>Update date</th>
12 </tr>
13 {% for report in crashreports %}
14 <tr>
15   <td>{{ report.appVersion }}</td>
16   <td>{{ report.customNote }}</td>
17   <td>{{ report.debugMode }}</td>
18   <td>{{ report.deviceModel }}</td>
19   <td><a href="/detail/{{ report.key.id() }}">link</a></td>
20   <td>{{ report.os }}</td>
21   <td>{{ report.osVersion }}</td>
22   <td>{{ report.uniqueDeviceIdentifier }}</td>
23   <td>{{ report.updateDate }}</td>
24 </tr>
25 {% endfor %}
26 </table>

```

Last 48h reports: 0

Last 48h Android reports: 0

Last 48h IOS reports: 0

Last 48h Windows Phone reports: 0

Total reports: 160

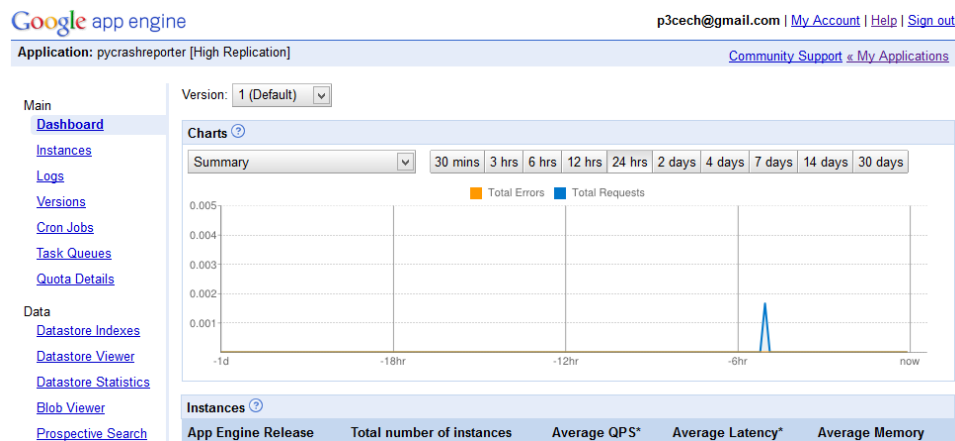
App version	Custom note	Debug mode	Device model	Report	OS	OS version	Unique Device Identifier	Update date
1.0		True	iPad mini 2G (WiFi)	link	ios	7.1		2014-04-19 16:51:38.493120
1.0.0.0		False	NOKIARM-914_eu_hungary_422	link	wp	8.0.10517.0		2014-04-19 16:51:00.767610
1.0		True	ZTE Blade	link	android	19		2014-04-19 16:50:28.567710
1.0		True	iPad mini 2G (WiFi)	link	ios	7.1		2014-04-17 19:01:46.613330
1.0.0.0		False	NOKIARM-914_eu_hungary_422	link	wp	8.0.10517.0		2014-04-17 19:01:15.720690
1.0.0.0		False	NOKIARM-914_eu_hungary_422	link	wp	8.0.10517.0		2014-04-16 05:47:33.455390
1.0.0.0		False	NOKIARM-914_eu_hungary_422	link	wp	8.0.10517.0		2014-04-16 05:45:49.290020
1.0.0.0		False	NOKIARM-914_eu_hungary_422	link	wp	8.0.10517.0		2014-04-16 05:45:39.388120
1.0.0.0		False	NOKIARM-914_eu_hungary_422	link	wp	8.0.10517.0		2014-04-16 05:44:37.409660
1.0.0.0		False	NOKIARM-914_eu_hungary_422	link	wp	8.0.10517.0		2014-04-16 05:42:28.682930
1.0.0.0		True	NOKIARM-914_eu_hungary_422	link	wp	8.0.10517.0		2014-04-16 05:33:10.854310
1.0.0.0		True	NOKIARM-914_eu_hungary_422	link	wp	8.0.10517.0		2014-04-16 05:33:01.177400

Obr. 29. Statistika se seznamem reportů

8.7 Webová správa serverové aplikace

GAE aplikace disponuje relativně rozsáhlou správou na webovém rozhraní. Pod tuto správu patří například nastavení plateb, přehled limitů, správa datových úložišť, výpis logů, nastavení služby cron nebo všeobecný přehled o aplikaci. [15]

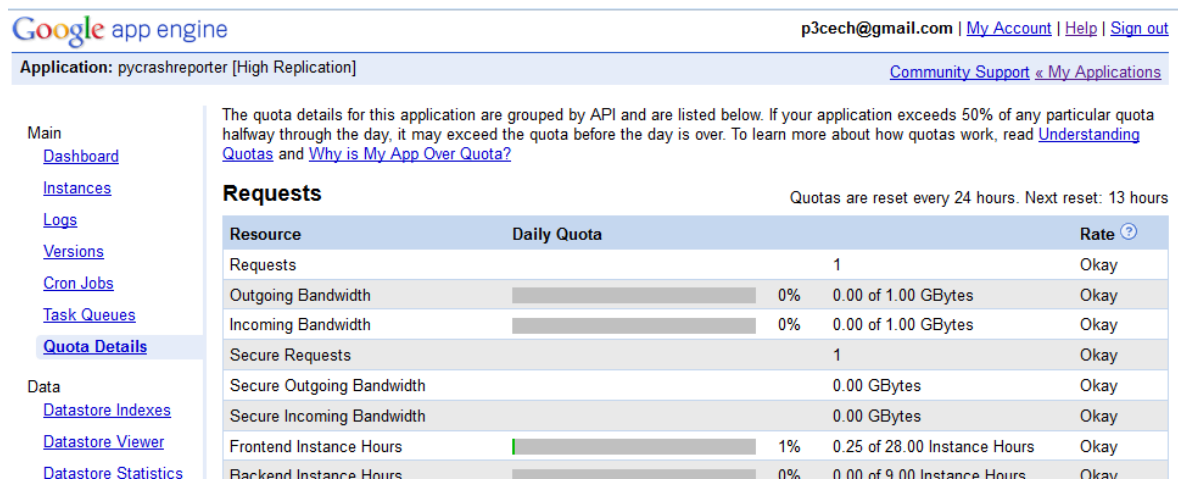
8.7.1 Všeobecný přehled



Obr. 30. Záložka Dashboard ve webovém rozhraní GAE

V záložce „Dashboard“ je možné sledovat vytížení aplikace v reálném čase. Dále je tu přehled využití limitů, poslední volané URL a počet chybových kódů (4XX a 5XX).

8.7.2 Přehled limitů



Obr. 31. Přehled limitů GAE

Záložka s přehledem limitů poskytuje ucelený přehled o procentuálním využitím limitů, které má GAE aplikace.

8.7.3 Správa Datastore

Google app engine p3cech@gmail.com | [My Account](#) | [Help](#) | [Sign out](#)

Application: pycrashreporter [High Replication] [Community Support](#) < [My Applications](#)

Main

- [Dashboard](#)
- [Instances](#)
- [Logs](#)
- [Versions](#)
- [Cron Jobs](#)
- [Task Queues](#)
- [Quota Details](#)

Data

- [Datastore Indexes](#)
- [Datastore Viewer](#)
- [Datastore Statistics](#)
- [Blob Viewer](#)
- [Prospective Search](#)
- [Text Search](#)
- [Datastore Admin](#)
- [Memcache Viewer](#)

Administration

- [Application Settings](#)
- [Permissions](#)
- [Blacklist](#)
- [Admin Logs](#)

Query Create

By kind: CrashReport kinds as of 0:00:00 ago Number of Columns to Display: 100

[Options](#)

By GQL:

SELECT * FROM CrashReport

Learn more about [GQL syntax](#).

CrashReport Entities

< Prev 20 1-20 Next 20 >

<input type="checkbox"/>	ID/Name	appVersion	customNote	debugMode	deviceModel	os	osVersion	report
<input type="checkbox"/>	id=4546618019807232	1.0		True	ZTE Blade	android	19	{REPORT_ID=f670a42b-34f7-4ff-9a9e-306d11ce1d08, APP_VERSION_CODE=1, APP_VERSION_NAME=1.0, PACKAGE_NAME=com.fai.test1.app, FILE_PATH=/data /data/com.fai.test1.app/files, PHONE_MODEL=Blade, ANDROID_VERSION=4.4.2, BUILD=BOARD=blade BOOTLOADER=unknown...
<input type="checkbox"/>	id=4644337115725824			True				test report content

Obr. 32. Správa databáze Datastore ve webovém rozhraní GAE

Přímo na webovém rozhraní aplikace GAE lze procházet záznamy, v tomto případě reporty. Lze dokonce využít dotazování pomocí jazyka GQL, který je obdobou SQL jazyka.

9 MOŽNOSTI ROZŠÍŘENÍ

GAE má značné omezení. Verze zdarma bude dostačující pro mobilní aplikace s omezeným počtem uživatelů, nebo pro aplikace s malým množstvím pádů. Množství pádů však nejde dopředu předvídat, ale lze očekávat podle řádového počtu aktivních uživatelů. Proto je u každé aplikace nutné si uvědomit, zda při navýšení počtu uživatelů bude přijatelné si připlatit za rozšíření kritických limitů GAE. Je mnoho kvalitních řešení třetích stran, které lze využít, ale ne každému nabízená řešení vyhovují. Proto byla vytvořena tato práce, aby nabídla možnosti vlastního způsobu zachytávání pádů na majoritních mobilních platformách a navrhnout základ vlastního řešení na platformě GAE.

9.1 Přidání nových parametrů

Protože vytvořená GAE aplikace využívá pro uložení reportů úložiště Datastore, je přidání, nebo úprava parametrů reportu jednoduchá. Datastore úložiště je totiž soubor entit a každá entita v tomto úložišti může mít různé parametry. Stejně tak jednoduché je rozšíření systému o další mobilní platformy. Přidání další mobilní platformy do systému spočívá v implementaci zachytávání pádů a odesílání reportů ve stanoveném tvaru na klientské části. Serverová část tak bude funkční pro novou platformu i bez jakékoli úpravy. [15]

ZÁVĚR

Cílem této práce bylo vytvoření systému pro zachytávání a zpracování reportů o pádu mobilních aplikací. Jako součást tohoto systému bylo navrženo řešení pro zachytávání a odesílání reportů na mobilních platformách Android, iOS a Windows Phone. Pro obsluhu reportů byla vytvořena serverová aplikace na platformě Google App Engine. V této práci byly použity programovací jazyky Java, Objective-C, C# a Python. Jako vývojová prostředí byla použita Android Studio, XCode, Visual Studio a Google App Engine Launcher.

Teoretická část práce zahrnuje rozbor obecných pojmů z oblasti zachytávání pádů mobilních aplikací, její součástí je i rozbor existujících řešení, zcela běžně používaných předními vývojáři mobilních aplikací. Obsahuje i popis, jakými způsoby lze obecně pády na majoritních mobilních systémech řešit. Je zde také rozebrán současný stav řešené problematiky, který není zcela vyhovující a tato práce může napomoci k lepšímu řešení daných problémů. Praktická část řeší zachytávání pádů na vybraných mobilních operačních systémech a odesílání reportů o těchto pádech na vlastní server. Vlastní server navržený na platformě GAE poté zachytává a zpracovává reporty. Součástí serverové části je ukládání reportů do databáze Datastore, notifikace vývojářů pomocí e-mailu a zobrazování jednoduchých statistik. Poslední část práce hodnotí jak aplikaci, tak i samotnou platformu GAE jako takovou, a to z hlediska bezpečnosti, spolehlivosti a možností jejího rozšíření.

Kód pro mobilní aplikace byl napsán tak, aby byl snadno znovupoužitelný při vytváření nové aplikace. Systém byl navržen i s ohledem na to, že v budoucnu nemusí být použita platforma GEA, ale jiný vlastní server. Při vytváření kódu pro obsluhu byly využity nejznámější a nejpoužívanější knihovny ve spojení s vlastní implementací odesílání dat ve vlastním formátu na vlastní server.

Serverová aplikace GAE byla navržena tak, aby byla funkční, použitelná a přesto dostatečně jednoduchá s ohledem na limity, které platforma má. Limity použité platformy lze dynamicky navyšovat po přechodu na prémiový účet pomocí poplatků. Vývojář by však musel zvážit, jestli je ochoten investovat peníze do rozšiřování těchto limitů, nebo bude chtít jiné vlastní řešení. Vývojáři mohou také využít některé z uvedených řešení třetích stran. Některá řešení jsou velice propracovaná a někdy dokonce zadarmo. U takových řešení se však neplatí peníze, ale vlastními daty, tedy informacemi, které jsou odesílány z mobilních zařízení.

Z hlediska bezpečnosti je systém založený na platformě GAE vyhovující. Google poskytuje pro své cloudové služby vysoké standardy zabezpečení dat a samotnou platformu, resp. společnost Google lze označit za důvěryhodnou. Navržený systém pro obsluhu pádů aplikací navíc není pro aplikace klíčový systém a v samotných reportech nejsou přenášena osobní údaje uživatelů aplikace, nebo jiná citlivá data.

Pro tuto práci byl použit verzovací systém na Google Drive a byly také uplatněny prvky projektového řízení při jejím rozvržení a sestavování. Pro projektové řízení byl použit systém Redmine. Ganttův diagram průběhu plnění úkolů je součástí jedné z příloh (P IV).

ZÁVĚR V ANGLIČTINĚ

The aim of this study was to develop a crash reporting system for mobile applications. As a part of this system it was designed the solution for capturing and sending reports on mobile platforms Android, iOS and Windows Phone. To operate the report it was created server applications on Google App Engine. In this work it was used programming languages Java, Objective- C, C # and Python. As a development environment was used Android Studio, XCode, Visual Studio and Google App Engine Launcher.

The theoretical part includes research with the general concepts of capturing mobile applications crashes. It also includes an analysis of the existing solutions that are routinely used by leading developers of mobile applications. In addition, it describes how crash reports can be handled on majority mobile operating systems. Briefly described the current situation is not entirely satisfactory, and the work should help find better solution for that. The practical part deals with crash reporting in selected mobile operating systems and reporting of these reports on its own server. Custom server platform designed to GAE then captures and processes the reports. GAE application server are handling saving reports to the Datastore, developers notification via e-mail and display simple statistics. The last part of the thesis evaluates both the application and the GAE platform itself as such in terms of reliability, security, and scalability.

Code for mobile applications was written so that it is easily reusable when creating new applications. The system was designed with regard to the fact that in the future may not be used GEA platform, but another own server. It was used most famous and most used library in conjunction with actual implementation of sending data in its own format on your own server, while creating code.

GAE Server application was designed to be functional, usable, and yet simple enough with regard to the limits that the platform has. The limits of designed platform can dynamically increase after upgrading to a premium account via taxes. The developer, however, before using the system must consider whether it is willing to invest money in expanding these limits or would want other custom solutions. Developers can also take advantage of some of mentioned third-party solutions. Some solutions are very sophisticated and sometimes even free. For such solutions, however, do not pay money, but the data itself, that is, information that are sent from mobile devices.

In terms of security, system based on GAE is suitable. Google provides high security standard for their cloud services and the platform itself, respectively Google can be described as trusted. Designed crash reporting system is not crucial for the application functionality and the reports themselves are not transmitted personal data of users or other sensitive data from the application.

For this work were used Google Drive version control system and were also utilized elements of project management in its layout and compilation. Redmine was used as project management system. Gantt diagram of tasks is part of one of the attachments (P IV).

SEZNAM POUŽITÉ LITERATURY

- [1] ACRA[LYZER]. *ACRA/acra* [online]. [cit. 2014-04-18]. Dostupné z: <https://github.com/ACRA/acra>
- [2] APPLE INC. *IOS Developer Library* [online]. [cit. 2014-04-19]. Dostupné z: <https://developer.apple.com/library/ios/>
- [3] BIT STADIUM GMBH. *HockeyApp - The Platform for Your Apps* [online]. 2014 [cit. 2014-04-18]. Dostupné z: <http://hockeyapp.net>
- [4] BURSTLY. *TestFlight; Beta Testing On The Fly* [online]. [cit. 2014-04-18]. Dostupné z: <http://testflightapp.com/>
- [5] CONDER, Shane a Lauren DARCEY. *Android App Publishing: Reading Android Market Crash Reports*. Tuts+ [online]. 2012 [cit. 2014-04-19]. Dostupné z: <http://code.tutsplus.com/tutorials/android-app-publishing-reading-android-market-crash-reports--mobile-9801>
- [6] *Crash reporter*. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2014-04-18]. Dostupné z: http://en.wikipedia.org/wiki/Crash_reporter
- [7] CRASHLYTICS. *The most powerful, yet lightest weight crash reporting solution for iOS and Android developers*. [online]. 2014 [cit. 2014-04-18]. Dostupné z: <http://try.crashlytics.com/>
- [8] CRITTERCISM, Inc. *Mobile Application Performance Management* [online]. 2013 [cit. 2014-04-18]. Dostupné z: <https://www.crittercism.com/>
- [9] Demystifying iOS Application Crash Logs. AZARPOUR, Soheil. *Ray Wenderlich* [online]. 2013 [cit. 2014-04-19]. Dostupné z: <http://www.raywenderlich.com/23704/demystifying-ios-application-crash-logs>
- [10] *Essential App Engine: Building High-Performance Java Apps with Google App Engine*. Indiana: Addison-Wesley, 2011. ISBN 9780132484756.
- [11] *Exploring iOS Crash Reports. Plausible Blog* [online]. 2013 [cit. 2014-04-19]. Dostupné z: <https://www.plausible.coop/blog/?p=176>
- [12] FALAFEL SOFTWARE. *Pro Windows Phone App Development*. 3. vyd. New York: Apress, 2013. ISBN 9781430247838.

- [13] FLURRY. *Flurry: Thrive in the New App Economy* [online]. 2014 [cit. 2014-04-18]. Dostupné z: <http://www.flurry.com/>
- [14] GALLAGHER, Matt. *Handling unhandled exceptions and signals*. GALLAGHER, Matt. *Cocoa with Love* [online]. 2008 [cit. 2014-04-18]. Dostupné z: <http://www.cocoawithlove.com/2010/05/handling-unhandled-exceptions-and.html>
- [15] GOOGLE Inc. *Google App Engine: Google Developers* [online]. [cit. 2014-04-18]. Dostupné z: <https://developers.google.com/appengine/>
- [16] GOOGLE INC. *Google on Android* [online]. 2013 [cit. 2014-04-27]. Dostupné z: <https://developers.google.com/android/>
- [17] GOOGLE INC. *Mobile App Analytics – Google Analytics* [online]. [cit. 2014-04-18]. Dostupné z: <https://www.google.com/analytics/mobile/>
- [18] HILLEGAS, HUNTER. *IOS CRASH REPORTING*. Foraker Labs [online]. 2012 [cit. 2014-04-18]. Dostupné z: <https://www.foraker.com/ios-crash-reporting/>
- [19] *How to get status code from webclient?*. Stackoverflow [online]. [cit. 2014-04-18]. Dostupné z: <http://stackoverflow.com/questions/3574659/how-to-get-status-code-from-webclient>
- [20] *IOS crash log catch, debug info.. Catch and send via email to the Dev team*. Stackoverflow [online]. [cit. 2014-04-18]. Dostupné z: <http://stackoverflow.com/questions/8233388/ios-crash-log-catch-debug-info-catch-and-send-via-email-to-the-dev-team>
- [21] KUMAR, Rob Napier and Mugunth. *IOS 6 Programming Pushing the Limits Advanced Application Development for Apple iPhone, iPad, and iPad Touch*. New York: Wiley, 2012. ISBN 978-111-8449-974.
- [22] LINDE, Andreas. *Live management of your crash reports for Mac OS X and iOS*. [online]. [cit. 2014-04-18]. Dostupné z: <http://quincykit.net/>
- [23] MICROSOFT. *MSDN Developer Network: Windows Phone* [online]. [cit. 2014-04-19]. Dostupné z: <http://msdn.microsoft.com/en-us/library/windowsphone>
- [24] *Overview of iOS Crash Reporting Tools: Part 1/2*. ROCCHI, Cesare. Ray Wenderlich | Tutorials for iPhone / iOS Developers and Gamers [online]. 2013 [cit. 2014-04-18]. Dostupné z: <http://www.raywenderlich.com/33669/overview-of-ios-crash-reporting-tools-part-1>

- [25] *Overview of iOS Crash Reporting Tools: Part 1/2*. ROCCHI, Cesare. Ray Wenderlich | Tutorials for iPhone / iOS Developers and Gamers [online]. 2013 [cit. 2014-04-18]. Dostupné z: <http://www.raywenderlich.com/34050/overview-of-ios-crash-reporting-tools-part-2>
- [26] PENNELL, Andy. *Error Reporting on Windows Phone 7*. MSDN Blogs [online]. 2010 [cit. 2014-04-18]. Dostupné z: <http://blogs.msdn.com/b/andypennell/archive/2010/11/01/error-reporting-on-windows-phone-7.aspx>
- [27] PLAUSIBLE LABS COOPERATIVE, Inc. *PLCrashReporter: In-process CrashReporter framework for iOS and Mac OS X* [online]. 2008 [cit. 2014-04-18]. Dostupné z: <https://www.plcrashreporter.org/>
- [28] *Programming Google App Engine*. Sebastopol: O'Reilly Media, Inc, 2010. ISBN 978-144-9383-039.
- [29] ROCHE, Xavier. *Replacing JNI Crashes by Exceptions on Android. Xavier roche's homework* [online]. 2014 [cit. 2014-04-20]. Dostupné z: <http://blog.httrack.com/blog/2013/08/23/catching-posix-signals-on-android/>
- [30] SALZ, Peggy Anne a Jennifer MORANZ. *The everything guide to mobile apps: a practical guide to affordable mobile app development for your business*. Avon, Mass.: Adams Media, c2013, 303 p. Everything series. ISBN 14-405-5533-8.
- [31] SATYA KOMATINENI, Dave MacLean a Eric Franchomme TECHNICAL REVIEWERS. *Pro Android 4*. New York: Apress, 2012. ISBN 978-143-0239-314.
- [32] SPLUNK COMPANY. *Crash Reports and Operational Intelligence for Android, iOS, Windows Phone & Windows 8 apps* [online]. [cit. 2014-04-18]. Dostupné z: <https://www.bugsense.com/>
- [33] WALES, Andrew. *5 Things You Should Be Doing With Google Mobile App Analytics Crash & Exception Measurement*. In: Google Analytics Blog [online]. [cit. 2014-04-18]. Dostupné z: <http://analytics.blogspot.cz/2013/02/5-things-you-should-be-doing-with.html>

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ACRA	Angl. Application Crash Reports for Android – knihovna pro zachytávání pádů na platformě Android
GAE	Angl. Google App Engine
HTTP	Angl. Hypertext Transfer Protocol – internetový protokol
HTTPS	Angl. Hypertext Transfer Protocol Secure – zabezpečená verze HTTP
JSON	Angl. JavaScript Object Notation – způsob zápisu dat
OTA	Angl.. Over The Air – označuje způsob distribuce aplikace
PHP	Hypertextový preprocesor; skriptovací programovací jazyk
POST	Dotazovací metoda protokolu HTTP
SDK	Angl.. Software Development Kit – sada softwarových vývojářských nástrojů
SQL	Angl. Structured Query Language – strukturovaný dotazovací jazyk
SSL	Angl. Secured Socked Layer – vrstva pro šifrované spojení
TLS	Angl. Transport Layer Security – následovník SSL
Wi-Fi	Označení pro několik standardů bezdrátové komunikace v počítačových sítích
WP	Angl. Windows Phone

SEZNAM OBRÁZKŮ

Obr. 1. Oznamovací dialog o pádu Android aplikace [5].....	12
Obr. 2. Android – odeslání reportu [5]	13
Obr. 3. Nastavení odesílání diagnostických dat v systému iOS 7.1	14
Obr. 4. Detail reportu o pádu v zařízení iOS 7.1	14
Obr. 5. Nastavení zpětné vazby v systému Windows Phone 8.1	15
Obr. 6. Zobrazení drobečků ve webovém rozhraní Crittecism [8]	18
Obr. 7. Zobrazení informace o odeslání reportu uživateli [1]	21
Obr. 8. Dialog s možností vložení poznámky uživatele k reportu [1].....	22
Obr. 9. Procházení reportů na Google play Developer Connsole (zdroj: http://www.codinguser.com/2013/02/dear-user-please-push-the-report- button/)	24
Obr. 10. Procházení reportů na iTunes Connect [9]	24
Obr. 11. Rozhraní Windows Phone Dev Center pro statistiky reportů.....	25
Obr. 12. Ukázka webového rozhraní Crittecism [8].....	26
Obr. 13. Ukázka webového rozhraní Bugsense (zdroj: https://plus.google.com/100427843958963974912/posts/AmjVJA7X7zB).....	27
Obr. 14. Ukázka webového rozhraní Crashlytics [7].....	28
Obr. 15. Ukázka přehledu reportů na webovém rozhraní Flurry [13]	29
Obr. 16. Diagram systému pro obsluhu pádů mobilních aplikací	30
Obr. 17. Propojení GAE s ostatním produkty Google [15]	31
Obr. 18. Ukázka použití vývojového prostředí Android Studio	34
Obr. 19. Testovací Android aplikace	35
Obr. 20. Ukázka použití vývojového prostředí XCode	36
Obr. 21. Testovací iOS aplikace	36
Obr. 22. Ukázka použití vývojového prostředí Visual Studio for Windows Phone 2012	37
Obr. 23 Testovací Windows Phone aplikace	38
Obr. 24. Zobrazení seznamu GAE aplikací	48
Obr. 25. Dvojitě ověřování Google účtu (Zdroj: http://www.lookyourstuff.com/2-step- verification-an-extra-security-layer-for-google-account/1133/)	49
Obr. 26. Vytvoření nové GAE aplikace [15].....	49
Obr. 27. Aplikace Google App Engine Launcher.....	50

Obr. 28. Označování notifikačních emailů s reporty	53
Obr. 29. Statistiky se seznamem reportů	54
Obr. 30. Záložka Dashboard ve webovém rozhraní GAE	55
Obr. 31. Přehled limitů GAE	55
Obr. 32. Správa databáze Datastore ve webovém rozhraní GAE	56

SEZNAM TABULEK

Tab. 1. Chybové signály operačních systémů.....	16
Tab. 2. Struktura odesílaných dat	42

SEZNAM PŘÍLOH

- P I Ukázka ACRA reportu o pádu aplikace pro operační systém Android
- P II Ukázka systémového reportu o pádu aplikace pro operační systém iOS
- P III Ukázka vlastního reportu o pádu aplikace pro operační systém Windows Phone
- P IV Ganttův diagram zpracování diplomové práce
- P V CD se zdrojovými kódy:
- Android aplikace napsané v jazyce Java a vytvořené v Android Studio s implementovaným zachytáváním a odesíláním vlastních reportů pomocí knihovny ACRA
 - iOS aplikace napsané v jazyce Objective-C, vytvořené v prostředí XCode s implementovaným zachytáváním a odesíláním vlastních reportů pomocí knihovny PLCrashReporter
 - Windows Phone aplikace napsaná v jazyce C#, vytvořené ve Visual Studio Express 2012 for Windows Phone s implementovaným zachytáváním vlastních reportů
 - GAE aplikace napsaná v jazyce Python, obsluhující reporty včetně ukládání reportů do úložiště Datastore, odesílání e-mailových notifikací, autentizace a zobrazování statistik z uložených reportů

PŘÍLOHA P I: UKÁZKA ACRA REPORTU O PÁDU APLIKACE PRO OPERAČNÍ SYSTÉM ANDROID

```
1  {REPORT_ID=8516fe13-6083-4bf0-ad40-697a1ba1f0f5,
  APP_VERSION_CODE=1, APP_VERSION_NAME=1.0,
  PACKAGE_NAME=com.fai.test1.app,
  FILE_PATH=/data/data/com.fai.test1.app/files, PHONE_MODEL=Blade,
  ANDROID_VERSION=4.4.2, BUILD=BOARD=blade
2  BOOTLOADER=unknown
3  BRAND=ZTE
4  CPU_ABI=armeabi
5  CPU_ABI2=armeabi-v6l
6  DEVICE=blade
7  DISPLAY=cm_blade-userdebug 4.4.2 KOT49H f8d3f48808 test-keys
8  FINGERPRINT=ZTE/N880E_JB4_2/atlas40:4.2/JOP40C/20121121.110335:user
  /release-keys
9  HARDWARE=blade
10 HOST=user-PA65-UD3-B3
11 ID=JRO03C
12 IS_DEBUGGABLE=true
13 MANUFACTURER=ZTE
14 MODEL=Blade
15 PRODUCT=blade
16 RADIO=unknown
17 SERIAL=unknown
18 TAGS=test-keys
19 TIME=1391079193000
20 TYPE=userdebug
21 UNKNOWN=unknown
22 USER=zeelog
23 VERSION.CODENAME=REL
24 VERSION.INCREMENTAL=f8d3f48808
25 VERSION.RELEASE=4.4.2
26 VERSION.RESOURCES_SDK_INT=19
27 VERSION.SDK=19
28 VERSION.SDK_INT=19
29  , BRAND=ZTE, PRODUCT=blade, TOTAL_MEM_SIZE=170393600,
  AVAILABLE_MEM_SIZE=27705344, CUSTOM_DATA=,
  STACK_TRACE=java.lang.RuntimeException: This is a crash
30  at com.fai.test1.app.MainActivity$1.onClick(MainActivity.java:27)
31  at android.view.View.performClick(View.java:4445)
32  at android.view.View$PerformClick.run(View.java:18429)
33  at android.os.Handler.handleCallback(Handler.java:733)
34  at android.os.Handler.dispatchMessage(Handler.java:95)
35  at android.os.Looper.loop(Looper.java:136)
36  at android.app.ActivityThread.main(ActivityThread.java:5114)
37  at java.lang.reflect.Method.invokeNative(Native Method)
38  at java.lang.reflect.Method.invoke(Method.java:515)
39  at
  com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteIn
  it.java:781)
40  at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:597)
41  at dalvik.system.NativeStart.main(Native Method)
42  , INITIAL_CONFIGURATION=compatScreenHeightDp=508
43  compatScreenWidthDp=320
44  compatSmallestScreenWidthDp=320
45  customTheme=system
46  densityDpi=240
47  ...
```

```
48 0.rectSize=[0,0,480,800]
49 0.refreshRate=61.000004
50 0.rotation=ROTATION_0
51 0.getSize=[480,800]
52 0.width=480
53 USER_APP_START_DATE=2014-04-19T18:50:21.000+02:00,
  USER_CRASH_DATE=2014-04-19T18:50:24.000+02:00,
  DUMPSYS_MEMINFO=Permission Denial: can't dump meminfo from from
  pid=1933, uid=10059 without permission android.permission.DUMP
54 ...
55 WIFI_COUNTRY_CODE=cz
56 WIFI_DISPLAY_ON=0
57 WIFI_NETWORKS_AVAILABLE_NOTIFICATION_ON=1
58 WIFI_ON=1
59 WIFI_SCAN_ALWAYS_AVAILABLE=0
60 WIFI_SLEEP_POLICY=2
61 WIFI_SUSPEND_OPTIMIZATIONS_ENABLED=1
62 WIFI_WATCHDOG_ON=1
63 WIRELESS_CHARGING_STARTED_SOUND=/system/media/audio/ui/WirelessChar
  gingStarted.ogg
64 , SHARED_PREFERENCES=default.acra.lastVersionNr=1
65 }
```

PŘÍLOHA P II: UKÁZKA SYSTÉMOVÉHO REPORTU O PÁDU APLIKACE PRO OPERAČNÍ SYSTÉM IOS

```
1 Incident Identifier: 5CFA3F93-BEFA-4CFD-9FC1-9EF86B7E3F7D
2 CrashReporter Key:   TODO
3 Hardware Model:      iPad4,4
4 Process:             Test1 [376]
5 Path:                /var/mobile/Applications/3FC0B7EA-8B05-46BC-9CC8-
6                       06F0ECFB20AC/Test1.app/Test1
7 Identifier:          com.fai.Test1
8 Version:             1.0
9 Code Type:           ARM-64
10 Parent Process:     launchd [1]

11 Date/Time:          2014-04-19 16:51:23 +0000
12 OS Version:         iPhone OS 7.1 (11D167)
13 Report Version:     104

14 Exception Type:     SIGSEGV
15 Exception Codes:    SEGV_ACCERR at 0x0
16 Crashed Thread:    0

17 Thread 0 Crashed:
18 0   Test1                0x00000001000b6410
19   0x1000b0000 + 25616
20 1   UIKit                0x000000018b4690c8
21   0x18b420000 + 299208
22 2   UIKit                0x000000018b46905c
23   0x18b420000 + 299100
24 3   UIKit                0x000000018b452538
25   0x18b420000 + 206136
26 4   UIKit                0x000000018b468a5c
27   0x18b420000 + 297564
28 5   UIKit                0x000000018b4686f0
29   0x18b420000 + 296688
30 6   UIKit                0x000000018b463388
31   0x18b420000 + 275336
32 7   UIKit                0x000000018b434b68
33   0x18b420000 + 84840
34 8   UIKit                0x000000018b432c58
35   0x18b420000 + 76888
36 ...

37 Thread 1:
38 0   libsystem_kernel.dylib 0x00000001953f5aa8
39   0x1953f4000 + 6824
40 1   libdispatch.dylib      0x00000001952f999c
41   0x1952f4000 + 22940

42 Thread 2:
43 0   libsystem_kernel.dylib 0x000000019540ee74
44   0x1953f4000 + 110196
45 1   libsystem_pthread.dylib 0x000000019548d54c
46   0x19548c000 + 5452
47 ...
48 Binary Images:
49 0x1000b0000 - 0x1000dbfff +Test1 arm64
50 <91a907e35b9f32ec818d121e9d99671b>
```

```
/var/mobile/Applications/3FC0B7EA-8B05-46BC-9CC8-  
06F0ECFB20AC/Test1.app/Test1  
36 0x186e10000 - 0x186e17fff AccessibilitySettingsLoader arm64  
/System/Library/AccessibilityBundles/AccessibilitySettingsLoader.bu  
ndle/AccessibilitySettingsLoader  
37 0x1870d4000 - 0x1871f7fff AVFoundation arm64  
<71b3ba6895883ad3bec5a25d26af998e>  
/System/Library/Frameworks/AVFoundation.framework/AVFoundation
```

PŘÍLOHA P III: UKÁZKA VLASTNÍHO REPORTU O PÁDU

APLIKACE PRO OPERAČNÍ SYSTÉM WINDOWS PHONE

```
1 at Test1.MainPage.Button_Click(Object sender, RoutedEventArgs e)
2 at System.Windows.Controls.Primitives.ButtonBase.OnClick()
3 at System.Windows.Controls.Button.OnClick()
4 at
  System.Windows.Controls.Primitives.ButtonBase.OnMouseLeftButtonUp(MouseButtonEventArgs e)
5 at System.Windows.Controls.Control.OnMouseLeftButtonUp(Control ctrl, EventArgs e)
6 at MS.Internal.JoltHelper.FireEvent(IntPtr unmanagedObj, IntPtr unmanagedObjArgs, Int32 argsTypeIndex, Int32 actualArgsTypeIndex, String eventName)
```

PŘÍLOHA P IV: GANTTŮV DIAGRAM ZPRACOVÁNÍ DIPLOMOVÉ PRÁCE

