VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

SYSTÉM PRO SLEDOVÁNÍ VYUŽITÍ MOBILNÍCH APLIKACÍ

SYSTEM TO MONITOR APPLICATION USAGE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MAREK NEVŘELA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. RADEK KOČÍ, Ph.D.**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Nevřela Marek, Bc.**
Obor: Inteligentní systémy
Téma: **Systém pro sledování využití mobilních aplikací**
System to Monitor Application Usage
Kategorie: Softwarové inženýrství

Pokyny:

1. Prostudujte problematiku tvorby mobilních aplikací a sledování běhu aplikací.
2. Proveďte analýzu existujících nástrojů pro sledování běhu aplikací, vyhodnoťte jejich výhody, nevýhody, možnosti nasazení a omezení.
3. Navrhněte systém pro sledování běhu mobilních aplikací a jejich použití uživatelem. Zaměřte se na sledování posloupnosti kroků v mobilní aplikaci, přístupů do vybraných částí aplikace a využívání aplikace v závislosti na denní době.
4. Systém bude mít několik částí. Mobilní část implementujte pro operační systém Android. Sběr a vyhodnocení dat implementujte jako serverovou aplikaci. Zobrazení výsledných dat bude mít na starosti klientská aplikace.
5. Připravte vhodnou testovací sadu a vyhodnoťte splnění požadavků na systém.

Literatura:

- Tutorialspoint. Android Tutorial. http://www.tutorialspoint.com/android/android_tutorial.pdf, dostupné říjen 2016.
- Mark L. Murphy. The Busy Coder's Guide to Android Development. https://commonsware.com/Android/Android_3-6-CC.pdf, dostupné říjen 2016.

Při obhajobě semestrální části projektu je požadováno:

- První tři body zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kočí Radek, Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Diplomová práce popisuje vývoj systému pro sledování používání mobilních aplikací na platformě Android. Cílem je vytvořit systém, který automatizovaně sbírá data v pozadí mobilních aplikací a je schopen je analyzovat. Práce se zabývá analýzou stávajících systémů a na jejím základě jsou navrženy požadavky na systém. Dále je popsán návrh celého systému pro sledování mobilních aplikací. V jednotlivých kapitolách je podrobně popsán návrh a implementace mobilní části, desktopové aplikace a serverové části systému a komunikace mezi nimi.

Abstract

Master's thesis describes development of the system which monitores a mobile application usage on the Android platform. The aim is to create the system which collects data in the background of a mobile application automatically and is able to analyze them. The thesis analyses the existing systems and proposes requirements for the new system based on them. Design of the whole system to monitor mobile application usage is the next part of the thesis. Design and implementation of mobile part, desktop application and server part of the systems and communication between them are described in the individual chapters.

Klíčová slova

Android, sledování mobilních aplikací, analýza uživatelského prožitku, mobilní aplikace, desktopová aplikace, JavaFX

Keywords

Android, monitoring mobile applications, analysis user experience, mobile application, desktop application, JavaFX

Citace

NEVŘELA, Marek. *Systém pro sledování využití mobilních aplikací.* Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Kočí, Ph.D.

Systém pro sledování využití mobilních aplikací

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Radka Kočího, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Marek Nevřela
23. května 2017

Poděkování

Rád bych poděkoval vedoucímu práce, Ing. Radku Kočímu, Ph.D., za cenné rady, připomínky a ochotu při tvorbě technické zprávy a návrhu systému.

Obsah

Kapitola 1

Úvod

Informační technologie jsou v současné době jedním z nejrozšířenějších technických odvětví na světě. Setkáváme se s nimi prakticky všude – doma, v práci, na cestách, protože zasahují do všech odvětví průmyslu. Mezi aktuální trendy patří chytrá zařízení, ať už se jedná o mobilní telefony, tablety, televize nebo nositelnou elektroniku. Mobilní telefon má svou pozici na trhu již několik let jistou, a stále ji posiluje. Oproti tomu doplňková elektronika, například v podobě chytrých hodinek, své uplatnění stále hledá. Mobilní telefon, jehož původní hlavní funkcí byla bezdrátová hlasová komunikace s ostatními telefony, se mění. V dnešní době se využívá kromě samotné komunikace také k přístupu k sociálním sítím, fotografování, poslechu hudby, sledování filmů, vyhledávání informací nebo ovládání domácnosti.

V současnosti vlastní svůj chytrý telefon přes 2,6 miliardy lidí po celém světě, což přesahuje třetinu světové populace a jejich počet stále roste [9]. Rozdílem mezi obyčejným mobilním telefonem a chytrým telefonem jsou právě aplikace. Aplikace jsou v dnešní době každodenní součástí všech uživatelů, kteří svůj chytrý telefon používají. Důležitým prvkem v těchto aplikacích je neustálá konektivita. Většina dnešních aplikací nějakým způsobem ke svému chování využívá přístupu k internetu a jejich cílem je zjednodušit uživateli přístup ke vzdáleným datům. Lidé tráví 90 % času práce s telefonem v aplikacích a jen 10 % využíváním prohlížeče. Uživatelé navíc začínají upřednostňovat využívání chytrých zařízení se specializovanými aplikacemi před stolními počítači nebo notebooky, kde je běžnější využití právě internetových prohlížečů [5].

Mobilní aplikace tak mají rok od roku silnější pozici na trhu a s tímto se zvyšují požadavky na konkrétní aplikace a také vývojáře. Kromě samotného vývoje je však také důležitý marketing, protože aplikací neustále přibývá a tak se zvětšuje i konkurence. Pokud má být aplikace úspěšná, tak musí být správně propagována, aby získala nové uživatele. Zároveň však je nutné, aby si uchovávala svoje stávající klienty. K tomu je nutné zjišťovat jejich zpětnou vazbu - jak jsou s aplikací spokojeni nebo které věci na aplikaci jim vadí. Tento způsob je však ze strany uživatelů dobrovolný a množství zpětné vazby většinou není dostatečné. Aby aplikace udržovala krok s konkurencí, je nutné pochopit, jakým způsobem uživatelé při práci s aplikací přemýšlejí a tyto znalosti využít k dalšímu vylepšení aplikace, aby byli uživatelé spokojeni.

Cílem práce je vytvořit systém pro sledování chování uživatelů v mobilních aplikacích, který by umožnil úspěšný rozvoj těchto aplikací. Tento systém, složený z několika částí, bude sbírat data o používání konkrétní mobilní aplikace. Data budou centralizovaně ukládána v serverových databázích. Vývojář aplikace bude využívat další část systému – klientský program, který umožní zobrazit data ze serveru takovým způsobem, aby vypovídaly o způ-

sobu použití aplikace. Systém bude pracovat s aplikacemi OS Android, který je momentálně rozšířen na 87,5 % všech chytrých mobilních telefonů [6].

Ve druhé kapitole diplomové práci jsou analyzována existující řešení a požadavky na systém a je také představena celková architektura systému. Ve třetí kapitole je popsán modul knihovny, který zajišťuje sběr dat v konkrétních aplikacích a odesílá je na server. Čtvrtá kapitola je zaměřena na desktopovou aplikaci a obsahuje návrh, výběr platformy i použité technologie při tvorbě aplikace. Součástí je také popis implementace této částí systému. V rámci páté kapitoly je detailně popsána serverová část systému, návrh databázové části, způsob práce s uloženými daty a implementační detaily jednotlivých funkcí. Šestá kapitola se věnuje testování aplikace a vyhodnocení splnění požadavků na systém. Pro tyto účely byla navržena a implementována jednoduchá testovací aplikace. V poslední kapitole se nachází závěrečné shrnutí a další možnosti rozšíření systému.

Kapitola 2

Analýza a návrh

Tato kapitola popisuje vybrané existující systémy, jejichž cílem je monitorovat chování uživatelů v mobilních aplikacích. V našem rozboru jsme se soustředili pouze na systémy pracující s OS Android, což je mobilní platforma, pro kterou je směřován vývoj sledovacího systému. V další části jsou specifikovány požadavky na funkčnost systému a popsán návrh struktury celého systému.

2.1 Existující systémy

V této části budou popsány již existující systémy pro sledování činnosti uživatelů v mobilních aplikacích. Analýza bude zaměřena zejména na jejich funkcionalitu, využití v praxi a způsob prezentace výsledků uživateli systému. V závěru budou zhodnoceny finanční nároky analyzovaných systémů a zdůrazněny nevýhody při jejich použití.

Jelikož mobilní aplikace dostávají s postupem času čím dál tím větší prostor v porovnání se softwarem pro osobní počítače, analýza a studium uživatelského prožitku je nyní již nutností. V současnosti již existuje mnoho nástrojů, které se zabývají analýzou chování uživatelů. Jako reprezentativní zástupci systémů tohoto typu byly vybrány 3 systémy – Appsee [1], CleverTap [2] a Firebase Analytics [3] od společnosti Google.

Appsee

Tato aplikace byla vybrána díky své široké škále funkcí. Postupně budou analyzovány některé z nich.

Základní funkcí všech systémů je sbíraní dat o uživateli – jak často aplikaci používá, jak dlouho pracuje s aplikací denně nebo týdně nebo v jakém čase ji obvykle používá. Sesbíraná data lze poté segmentovat do určitých intervalů a například zjistit, jak je závislé použití aplikace na denní době. Tato funkce je velmi užitečná pro základní představu, jaké je využití aplikace u cílového uživatele. Kromě těchto dat pracujících s veličinou času jsou zaznamenávány například počet spuštění aplikace nebo počet unikátních přístupů. Tato data jsou zobrazeny v tabulkách nebo sloupcových grafech.

Další základní funkcí, kterou poskytuje téměř každý software je funkce *funnels*[1]. Tato funkce umožní například zjistit, jaký počet nebo procentuální zastoupení uživatelů vstoupilo na přihlašovací obrazovku a poté se v aplikaci přesunulo na obrazovku obnovení hesla. Tímto je umožněno zjistit, nakolik je využívaná určitá část aplikace – zejména ty obrazovky, pro

[1] Funnels – výlevka, trychtýř (volně přeloženo), neexistuje odpovídající český ekvivalent charakterizující tuto funkci. Pro další pojmenování této funkce bude používáno označení filtrování.

které je specifická jedna funkce, kterou poskytují. Pro zobrazení je využit speciální diagram s textovými popisky, ale dostačující by byl i samotný textový popis.

Podobnou funkcí je sledování vlastních událostí. Tím je myšlena možnost sledování aplikace dokud nedojde do určitého stavu nebo neproběhne zadaná událost. Tímto je například dohrání všech úrovní nebo uplatnění bonusových bodů v mobilní hře. Takto lze například upravovat složitost v případě hry. Informace jsou většinou zobrazeny v tabulce nebo grafu k porovnání s celkovým počtem uživatelů.

Sledování návratnosti uživatele do aplikace je další funkcí, která je poskytnuta systémem Appsee. Tato funkcionalita již není samozřejmostí každého systému. Pracuje s konkrétními unikátními přístupy do aplikace a zjišťuje, kolik uživatelů aplikaci opakovaně používá. Výsledná data jsou pak pro určité časové intervaly zobrazeny v tabulce.

Naopak mezi poměrně ojedinělé funkce systému patří nahrávání obrazovky uživatele během práce s aplikací. Shlédnutím této nahrávky lze jednoduše zjistit, jak konkrétní uživatel aplikaci používá. Tato funkce však není příliš použitelná pro 100 % uživatelů z těchto důvodů:

- Nahrávání obrazovky je další zátěží pro systém a tudíž dlouhodobě velkou měrou ovlivňuje spotřebu baterie.
- Ukládaní videí nutně potřebuje místo v paměti telefonu a při nahrávání kvalitního videa může obsadit relativně velkou část paměti.
- Nahraná videa je nutno postupně nahrávat na cloud nebo jiné úložiště pomocí internetu. To má za následek vyšší přenos dat z mobilního zařízení. a také rychlé zaplnění kapacity cloudu nebo jiného centrálního úložiště.
- Při ukládání záznamů může vývojář aplikace sledovat z časových důvodů pouze některé záznamy a absolutní většina zůstane nevyužita.

Zajímavou a poměrně jedinečnou funkcí je sběr dat pro vytvoření *heatmap*, která ukazuje jak často jsou používány jednotlivé oblasti přímo na konkrétním náhledu obrazovky a naopak které nejsou využívány vůbec.

Vhodnou funkcí pro analýzu chování aplikace je reportování jejich pádů. Takto lze získat data k situacím, které nenastanou při testování aplikace před jejím nasazením. Data pak jsou použity k odladění nalezených chyb.

CleverTap

Tento monitorovací systém obsahuje velmi podobné funkce jako Appsee – sbírá data o uživateli, využívá filtrování dat, vlastní události, sledování návratnosti. Naopak nepodporuje nahrávání video záznamu, zobrazení heatmap, ani reportování pádů aplikace. Tento software může být navíc kromě mobilních platforem využit na webových stránkách.

Kromě těchto funkcí pro analýzu aplikace je součástí systém pro oslovení určité skupiny uživatelů. Uživatelé jsou na základě svého chování přidělováni do skupin, které je charakterizují. Využít lze například různé typy notifikací ve webovém prohlížeči nebo aplikaci, případně SMS zprávy nebo e-mail. Notifikace mohou být odeslány uživateli okamžitě (např. oznámení o slevové akci), ale také na základě nějaké podmínky, kterou uživatel aplikace splní (např. objednání zboží na e-shopu). Tyto funkce jsou spíše určeny pro marketingové oddělení než pro vývojáře, nicméně jsou stále součástí analýzy a zlepšování uživatelského prožitku.

Firebase Analytics

Firebase Analytics je jedním z modulů obrovského systému Firebase. Ten byl vytvořen se záměrem pomoci při vývoji mobilních a webových aplikací. Samotný systém Analytics je poměrně nový systém[2] a funkčně je velmi podobný aplikacím Appsee a CleverTap. Nabízí sledování vlastních událostí, filtrování dat i sledování návratnosti.

Obrovskou výhodou této služby je, že každé zařízení je připojeno pomocí e-mailové adresy a uživatel je tak identifikován. Analytics tak má kromě základních informací, které posbírá aplikace, možnost využít dalších služeb Googlu – Gmailu nebo Google Play services. Z těchto dat lze pak dále vyčíst například pohlaví uživatele nebo z jaké země pochází. Samotný Analytics neposkytuje sledování pádů aplikace, ale k této analýze lze využít další integrovaný systémy Firebase - Crash Reporting. Obdobně pro zasílání notifikací poskytuje službu Firebase Notification. Kromě toho lze využívat využívat i další služby Firebase, které jsou vhodně umístěny ve stejném prostředí.

Zhodnocení

Aplikace z velké části odpovídají svými funkcemi základním požadavkům diplomové práce. Obecně jsou tyto systémy nastaveny takovým způsobem, aby je bylo možné vyzkoušet v omezené míře a poté se muselo za jejich služby platit.

Software Appsee poskytuje dvoutýdenní režim k vyzkoušení, poté je nutné za používání již platit. Systém CleverTap umožňuje výběr z několika úrovní balíčků. Nejmenší z nabízených možností je zdarma, avšak jeho některé funkce jsou omezené a jiné nejsou dostupné vůbec. Cena balíčků se odvíjí od množství událostí zapsaných při používání aplikací. Nejmenší placená verze balíčku je oceněna na 350 amerických dolarů. Posledním systémem je Firebase Analytics a jeho služby jsou v současné době zdarma[3]. Jediným omezením je limit typů událostí sledovaných v aplikaci, který je omezen 500 možnými druhy událostí. Pokud však uvažujeme, že tyto služby jsou u ostatních konkurentů placené, je možné předpokládat, že tyto služby nebudou v budoucnu zdarma a bez omezení, jak je tomu nyní. U těchto existujících systémů byly nalezeny tyto nedostatky:

- Každý měsíc se platí za poskytovanou službu, s přibývajícími koncovými uživateli je zvyšován poplatek
- Data jsou uložená u poskytovatele služby a není možnost s nimi pracovat jakýmkoliv způsobem, ale pouze pomocí poskytnutého API
- Systémy mají příliš obecné funkce a není možnost vytvořit vlastní grafy

2.2 Specifikace požadavků

Cílem této práce je vytvořit systém, který bude sloužit jako nástroj k analýze uživatelů mobilních aplikací, jejich způsobu uvažování při práci s aplikací a automatismů, ale také k dalším vylepšením při vývoji mobilních aplikací. Požadavky vycházejí z vlastních potřeb firmy Nextis s.r.o., které byly zjištěny během vývoje mobilních aplikací, ale také jsou inspirovány funkcemi již existujících systémů, které byly zmíněny v kapitole 2.1.

[2] Firebase Analytics byl představen v květnu 2016 na konferenci Google I/O
[3] Platné k datu 28. 12. 2016

Univerzálnost a snadná aplikovatelnost

Systém by měl být univerzální pro použití na jakékoliv mobilní aplikaci, která běží na OS Android. K použití bude potřeba vlastnit zdrojové kódy aplikace a umožnit k nim systému přístup. Systém provede analýzu zdrojových kódů a vloží systém pro monitorování chování uživatele do mobilní aplikace. Tímto procesem bude nutné projít pouze po úpravách zdrojových kódu před plánovaným vydáním.

Oddělitelnost verzí jednotlivých aplikací

V systému bude zaznamenáno každé aplikování systému na zdrojové kódy mobilní aplikace. Data, která budou sbírána, budou označena aktuální verzí aplikace a bude je tak později možno rozlišit, jednotlivě analyzovat nebo vzájemně porovnávat.

Sledování běhu mobilní aplikace

Systém v mobilní aplikaci bude monitorovat posloupnost jejího běhu. Každý významný krok uživatele a jím vyvolaná posloupnost akcí bude zaznamenána. Systém poté bude schopen vyhodnotit jednotlivé posloupnosti.

Sledování závislostí na různých stavech

Systém musí umět sledovat využívání aplikace v závislosti na různých stavech, například na denní době, připojení k bezdrátové síti nebo jazykové mutaci aplikace. Tyto informace umožní zjistit v jakém prostředí jsou aplikace používány nebo v jaké části dne.

Přehledné zobrazení informací

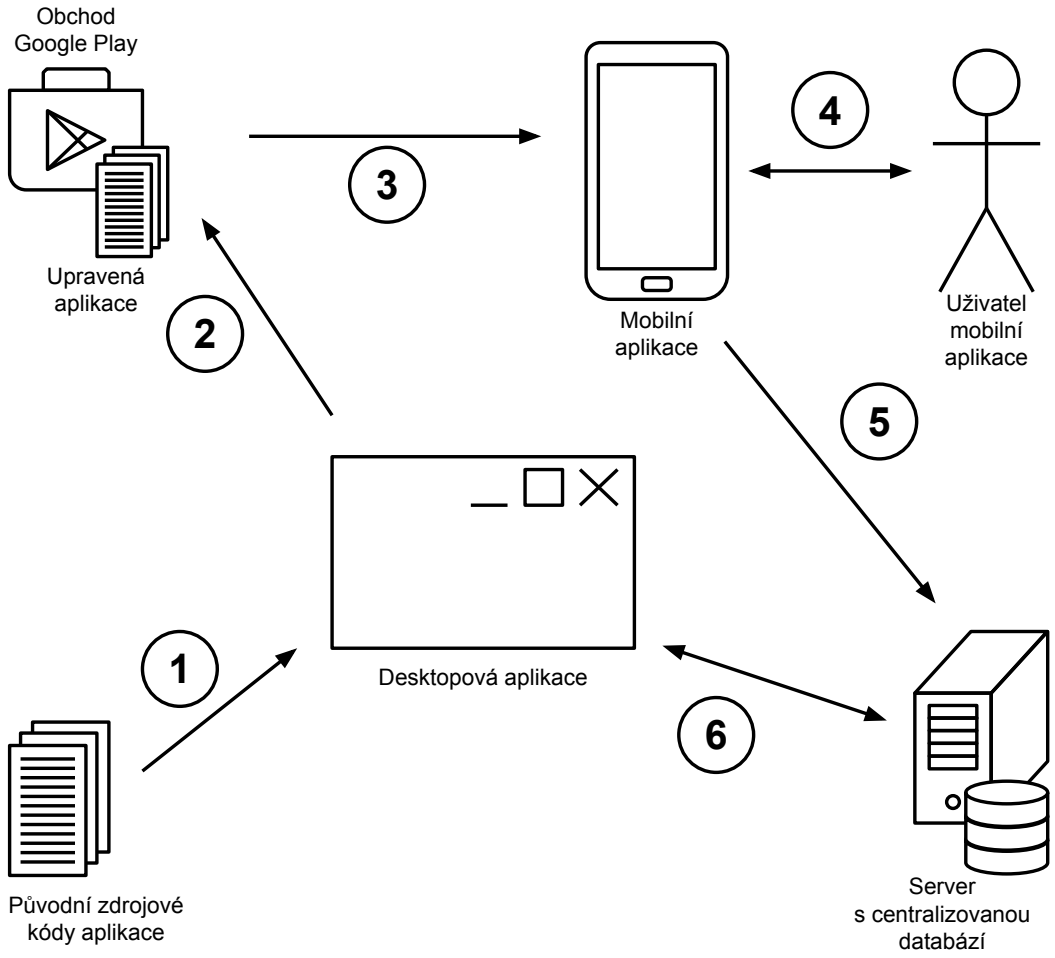
Data, která budou získaná z části systému vloženého do mobilní aplikace, budou zpracována a převedena do takové formy, aby poté mohly být zobrazeny pomocí grafů nebo tabulek podle svého zaměření. Tyto informace by měly být dle potřeby doplněny dalšími textovými popisky, aby jejich vypovídací hodnota byla co největší.

2.3 Návrh struktury systému

Abychom byli schopni analyzovat potřebná data získaná z mobilních zařízení, musíme tato data centralizovat. Proto je vhodné pro tento systém využít architekturu klient-server. Celý systém bude poměrně rozsáhlý, proto jej bylo nutné rozdělit do několika částí. Schéma zobrazené na obrázku 2.1 prezentuje celý běh systému od počátečního vložení části systému do mobilní aplikace až po zobrazení výsledných dat uživateli klientského programu. Jednotlivé kroky průběhu systému jsou popsány v odpovídajících částech.

1. Uživatel vloží zdrojové kódy své mobilní aplikace do desktopové aplikace systému. Ta provede jejich analýzu a importuje části potřebné pro sbíraní dat v mobilní aplikaci. Tento krok je využit pouze poprvé nebo pokud je nutné vydat novou verzi aplikace.

2. Upravená aplikace je publikována na Google Play nebo je dále distribuována alternativním způsobem. Tento krok, stejně jako předchozí, je proveden pouze při prvním vydání aplikace nebo její aktualizaci. Samotný proces distribuce aplikace není součástí systému. Tato procedura je řešena individuálně stranou, která využívá systém.



Obrázek 2.1: Schéma systému

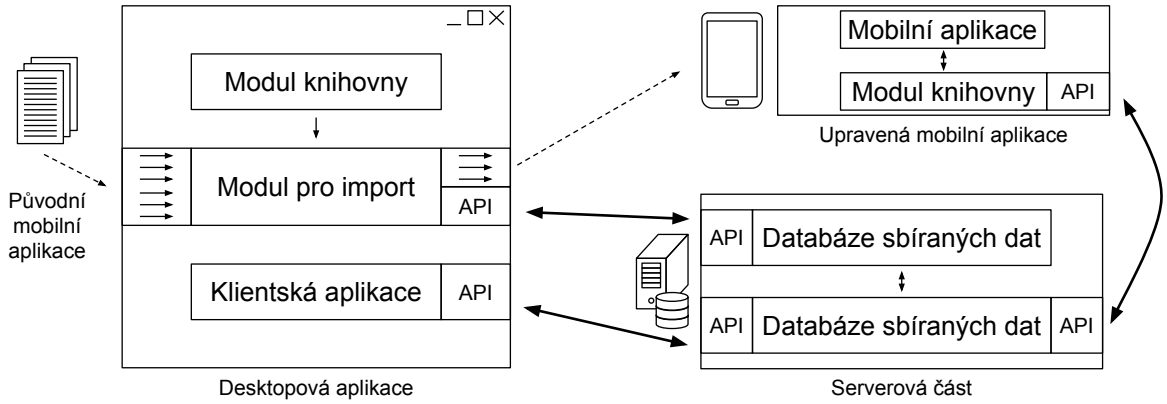
3. Potenciální uživatel mobilní aplikace stáhne a nainstaluje aplikaci do zařízení. Tento krok, stejně jako předchozí je součástí pouze procesu prvního vydání aplikace nebo její aktualizace.

4. Uživatel mobilní aplikace používá aplikaci obvyklým způsobem. Během používání jsou v pozadí zaznamenávány postupy, jak uživatel během práce s aplikací postupoval, které části navštívil nebo funkce, které využil. Tento proces probíhá neustále ve všech aplikacích, ve kterých byl již nasazen monitorovací systém.

5. Data, která byla nasbírána v mobilním zařízení, jsou hromadně odesílána na server skrze síť internetu. Na serveru jsou poté rozdělena a ukládána do příslušných tabulek. Stejně jako předchozí proces je i tento neustále opakován, aby nedocházelo k hromadění dat na mobilních zařízeních.

6. Desktopová aplikace zasílá požadavek na server, který zpracuje konkrétní data z tabulek databáze a odpoví zpět. Aplikace pak tyto informace pouze prezentuje, většinou v podobě grafů nebo tabulek. Tento proces je využíván pouze na základě interakce uživatele s aplikací, který chce zobrazit určitou informaci.

Systém tedy můžeme tímto rozdělit na tři části – modul knihovny, desktopovou aplikaci a serverovou část. Každá z těchto částí má vlastní rozhraní pro komunikaci. Schéma propojení jednotlivých částí je zobrazeno na obrázku 2.2. Desktopová aplikace obsahuje několik částí. Modul pro import, který je její součástí, zajišťuje import modulu knihovny do mobilní aplikace. Ta je poté její nedílnou součástí a sbírá informace o chování uživatele v aplikaci. Detailní návrh těchto částí systému bude podrobně popsán v následujících kapitolách 3, 4 a 5. Vytvoření plnohodnotné mobilní aplikace není součástí této práce. Pro otestování systému bude vytvořena vhodná aplikace, která pomůže otestovat správné chování systému a zároveň bude použita k vyhodnocení splnění požadavků na systém.



Obrázek 2.2: Schéma propojení jednotlivých části systému

Kapitola 3

Modul knihovny

Tato kapitola pojednává o knihovně určenou pro sběr dat v aplikacích využívající systém pro sledování chování uživatelů. Součástí kapitoly je návrh knihovny, návrh lokální databáze a implementace celého modulu.

Modul knihovny je soubor zdrojových kódů určených pro OS Android, který musí být součástí mobilní aplikace, která chce využívat tento systém. Jejím cílem je sbírat informace o práci uživatele v aplikaci a tyto informace odesílat na server. Pokud jsou tyto informace úspěšně uloženy do databáze umístěné na serveru, pak mohou být v zařízení smazány. Celý tento modul je obsažen v desktopové aplikaci, aby mohl být při importu systému vložen ke zdrojovým kódům aplikace.

3.1 Návrh knihovny

Nejdůležitější pro tuto knihovnu je fakt, aby při sbírání dat neměla žádný vliv na koncového uživatele aplikace. V praxi by totiž při používání této knihovny v jakékoliv aplikaci neměl být poznat rozdíl mezi aplikací s knihovnou sbírající data nebo bez ní. Stejně tak by knihovna v pozadí neměla výrazně využívat další zdroje zařízení – např. úložiště, paměť nebo baterii. S ohledem na tyto kritéria byl vytvořen návrh této knihovny.

Informací, které budou knihovnou zaznamenávány, bude poměrně velké množství, proto by nebylo výhodné zasílat každou informaci ihned při jejím zaznamenání. Data jsou nejprve ukládány v zařízení a později hromadně zasílány na server. To šetří zbytečné zatěžování mobilní sítě a tím i baterie zařízení. Knihovna je navržena tak, aby k její správné funkci nebylo potřeba trvalého přístupu k internetu. Pro odeslání dat na server tak postačí občasné připojení k internetu skrze síť WiFi, což by celkově mělo šetřit mobilní data. Jednotlivé komponenty knihovny sbírají vybrané události, při různých interakcích uživatele s aplikací. Tato část je detailněji popsána v sekci Komponenty. Systém umožňuje také sběr událostí, které zvolí uživatel systému. Tento proces je podrobněji popsán v sekci Vlastní události.

Sbírané informace se vždy váží ke konkrétnímu časovému okamžiku, ve kterém byly provedeny. Informace, které byly zaznamenány za sebou, již tvoří určitý časový úsek. Abychom byli schopni analyzovat chování uživatele v různých situacích, je nutné oddělit jednotlivé časové úseky a odpovídající seznam sesbíraných informací. Označme *session* jako časový úsek, který trvá od spuštění mobilní aplikace do jejího ukončení. Ke každé *session* je vázána posloupnost informací sesbíraných knihovnou při interakci s uživatelem během odpovídajícího časového úseku. Kromě těchto informací jsou se *session* pojena také statická data, jako

je používaný jazyk aplikace, její verze nebo vlastnosti zařízení (značka, verze OS, velikost, rozlišení obrazovky atd.)

3.1.1 Komponenty

Komponenty aplikace představují v této knihovně množinu tříd, které jsou zděděny od tříd často využívaných k vývoji aplikací. Tyto třídy pocházejí přímo z knihoven *Android*[1] nebo *Support Library*[2]. Použit je běžný model využívaný při vývoji mobilních aplikací pro OS Android, kdy je využívána při vývoji vlastních komponent jednoduchá dědičnost. Pokud tedy například chce vývojář vytvořit vlastní aktivitu, která bude zobrazovat určitá data, pak využijeme jednu z již implementovaných tříd z výše uvedených knihoven nebo jiných knihoven třetích stran. Vlastní aktivita pak bude pouze rozšířením (potomkem) původní třídy.

Třídy modulu knihovny jsou tedy potomky vybraných tříd z knihoven *Android* a *Support Library*. V těchto třídách jsou doplněny volání metod pro sběr dat. Sběr dat probíhá obvykle v částech kódu, kdy uživatel nějakým způsobem reagoval – jsou využity různé typy listenerů, volání specifických metod nebo metod životních cyklů. Tento způsob umožní zaznamenávání posloupnosti kroků v aplikaci sledováním aktivit, fragmentů nebo vybraných grafických komponent. Přehled sledovaných tříd a jejich akcí je zobrazen v tabulce 3.1.

Tabulka nám dle pořadí sloupců ukazuje objekty tříd, které jsou sledovány. Ke konkrétním objektům jsou zaznamenávány akce a stavy. Poslední sloupec nám ukazuje, kdy je daná událost vyvolána. V případě aktivit je často využit životní cyklus aktivity, případně změny informací o zařízení (v případě změny sítě nebo orientace zařízení). U grafických komponent jsou spouštěcí událostí většinou rozhraní konkrétních objektů nebo metody rodičovské třídy. U aktivit vždy sledujeme právě způsob připojení k internetu nebo orientaci zařízení, při kterém uživatel aplikaci používá. Kromě toho je zaznamenáván čas strávený na konkrétní aktivitě. V případě grafických komponent jsou sledovány akce nad těmito objekty – např. kliknutí, výběr položky z jeho nabídky nebo změnu stavu. Všechny tyto informace jsou zaznamenávány společně k akci, která byla provedena a vybraným vlastnostem daného objektu jako je jeho identifikátor, případně zobrazený text nebo jeho stav podle funkce, kterou zastává.

3.1.2 Vlastní události

Komponenty a jejich interakce s uživatelem nejsou jediný způsob, jak získávat data o chování uživatele v aplikaci. Pokud by byl systém omezen pouze na události sbírané vybranými komponentami, pak by byla celá analýza chování uživatele velmi strohá. Aby mohla být analýza i použití celého systému komplexní, je nutné uživateli umožnit sbírání dat, které on sám považuje za důležitá. K tomuto slouží vlastní události.

Vlastní událostí je myšlen moment, kdy probíhá specifická část zdrojového kódu – například zahájení stahování dat přes internet nebo spuštění vybrané písničky pro přehrávání. Pro tyto události lze jednoznačně nalézt ve zdrojovém kódu místo, kde by měly být zaznamenány. Kromě toho, že bude událost zaznamenána v systému, je také vhodné, aby bylo možné události stejného typu oddělit podle dalších kritérií – například při zahájení přehrávání hudebního alba může chtít uživatel systému zaznamenat, jaký je využíván mód přehrávání (náhodně zvolené písničky, opakování), délka alba a další informace. Z těchto

[1] https://developer.android.com/reference/

[2] https://developer.android.com/topic/libraries/support-library/

Třída	Zaznemaná akce	Zaznamenané hodnoty	Spouštěcí událost
Activity	show	wifi/mobile; port/land	zobrazení aktivity
Activity	hide	wifi/mobile; port/land	ukončení zobrazení aktivity
Activity	time	čas (s)	ukončení zobrazení aktivity
Activity	netChange	wifi/mobile/none	změna připojení k síti
Activity	orientationChange	port/land	změna orientace zařízení
EditText	textChange	id; text	při ztrátě zaměření
Button	click	id; text	při kliknutí
ImageButton	click	id	při kliknutí
ImageView	click	id	při kliknutí
TextView	click	id; text	při kliknutí
ScrollView	scroll	id; up/down	při ukončení skrolování
ViewPager	swipe	id; text; position	při přesunu na obrazovku
ListView	choose	id; text; position	při výběru ze seznamu
RadioButton	choose	id; text	při výběru prvku
Spinner	choose	id; text; position	při výběru ze seznamu
CheckBox	checkChange	id; text; checked	při změně zaškrtnutí
Switch	switch	id; text; checked	při změně stavu
ToggleButton	toggle	id; text; checked	při změně stavu

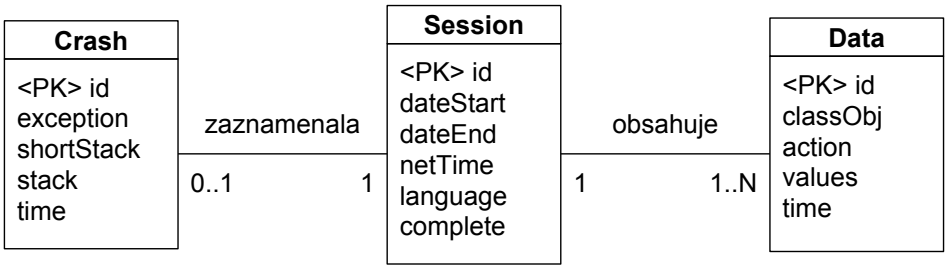
Tabulka 3.1: Sbírané události komponent OS Android

informací je poté možno porovnat, jaký typ nastavení uživatelé mobilní aplikace používají častěji.

K tomuto zaznamenání informací musí dojít co nejjednodušším způsobem, aby původní zdrojový kód byl co nejméně modifikován. Klíčový pro zaznamenávanou událost je její název, který je u každé události povinný. Další informace, které jsou k události vázány jsou zaznamenávány jako její parametry a všechny jsou v textové podobě. Pro zaznamenání události do systému je nutné do zdrojového kódu doplnit volání statické metody `event` třídy `LogEvent`, která je součástí knihovny. Uživatel systému nemusí kód zaznamenávající vlastní události psát ručně. Pro jejich zaznamenání je vhodné využít modul pro import knihovny, který umožňuje využít automatizovaný proces vložení – tato část je popsána v části 4.2.3.

3.1.3 Návrh lokální databáze

Pro ukládání dat v zařízení je použita lokální databáze typu SQLite, která je součástí knihovny. Databáze je složena z několika tabulek uchovávajících sesbíraná data a informace o jednotlivých *session*. ER diagram lokální databáze je vidět na obrázku 3.1. Entita *Session* obsahuje data vázané ke konktrétní *session* – např. čas začátku a ukončení, čistý čas používání aplikace během *session*, aktuálně používanou jazykovou mutaci aplikace a informaci o tom, zda je *session* označena jako již ukončená. Entita *Crash* obsahuje podrobnosti o pádech aplikace. Každý pád aplikace je vázán na konkrétní *session* a pokud k němu dojde, pak je celá *session* tímto ukončena. Třetí entita *Data* obsahuje již konkrétní data, která jsou sbírána během práce s aplikací. Zaznamenávány jsou informace o tom, kdy a jaká interakce byla provedena.



Obrázek 3.1: ER diagram lokální databáze pro modul knihovny

Statická data jako jsou model telefonu, verze OS nebo velikost obrazovky není nutné ukládat do databáze, protože jsou v každém časovém okamžiku používání aplikace stejné a nemění se. V databázové tabulce by tak byl vždy jen jeden záznam. Proto stačí tyto data získat při prvním použití aplikace a uložit si je k sdíleným preferencím aplikace (třída `SharedPreferences`). Tyto údaje jsou zobrazeny v tabulce 3.2 včetně jejich formátu zápisu. Výjimkou je informace o verzi operačního systému. Tento údaj je nutné získávat při každém započetí nové *session*, protože může docházet k upgrade OS. Tyto údaje pomohou zjistit zastoupení využívání konkrétních modelů zařízení, jejich vlastností a operačních systémů.

Označení	Typ dat	Poznámka
manufacturer	String	Výrobce
model	String	Model
ram	Number	Velikost RAM v MB
storage	Number	Velikost primárního úložiště v GB
screenSize	Number	Velikost obrazovky v palcích
screenResolution	Number, Number	Rozlišení obrazovky v px Width x Height
screenDensity	Number	Hustota pixelu v ppi

Tabulka 3.2: Statická data zařízení

3.2 Implementace knihovny

Tato kapitola pojednává o celkovém konceptu knihovny, jejím vývoji, sbírání dat pomocí upravených komponent i vlastních událostí. Součástí kapitoly je také část, která rozebírá datovou komunikaci mezi mobilním zařízením a serverem.

Knihovna byla implementována v programovacím jazyce Java. Podpora byla vybrána pro verze Android 4.1 Jelly Bean (verze API 16) a vyšší. Zařízení od této verze až po nejnovější verzi Android 7.1 Nougat (verze API 25) tvoří 98 % všech aktivních zařízení s operačním systémem Android [7].

3.2.1 Vývojové prostředí

Aplikace vyvíjené pro operační systém Android jsou ve většině případů vyvíjeny v programovacím jazyce Java. Pro vývoj je tedy nutné mít nainstalovaný Java Development Kit (JDK), který poskytuje základní nástroje pro vývoj v programovacím jazyce Java. Pro vývoj aplikací je vhodné použít vývojové prostředí (IDE), které by mělo urychlit práci. V současné době patří mezi nejpoužívanější Android Studio a Eclipse.

Pro vývoj knihovny bylo zvoleno Android Studio jako IDE doporučované tvůrci OS Android. Toto IDE je dostupné z webu [4]. Android Studio je založeno na IntelliJ IDEA, což je vývojové prostředí zaměřené zejména na programování v jazyce Java. Toto IDE nabízí jednoduché prvotní spuštění, včetně instalace Android SDK, buildovacího nástroje Gradle, emulátorů a obrazů systému či debuggeru. Dále jsou z něj jednoduše dostupné oficiální aktualizace všech nástrojů a obrazů systému poskytované přes Android SDK.

3.2.2 Sledování a sbírání událostí

Aby bylo možné konkrétní data od uživatelů analyzovat, musí se nejprve nasbírat a shromáždit. Sbírání dat, jak již bylo zmíněno v návrhu, můžeme rozdělit do dvou kategorií – grafické komponenty a aktivity.

Grafické komponenty sbírají data, která popisují nějakou reakci uživatele v aplikaci – např. kliknutí na tlačítko, výběr prvku ze seznamu nebo úprava textového pole. Pro tento druh sbíraní dat z komponent byly vybrány nejpoužívanější základní komponenty z knihoven *Android* a *Support Library*. Z těchto komponent byly vytvořeny specializované komponenty, které jsou rozšířením původních tříd. Tyto specializované třídy jsou doplněny podle primární funkce, kterou plní – např. třídy, které zaznamenávají akci kliknutí (`Button`, `ImageButton`, `ImageView`, `TextView`) mají překrytou metodu `performClick` třídy `View` tak, aby při akci kliknutí na objekt byla zapsána požadovaná data. Konkrétně pro třídy `Button` a `TextView` jsou navíc zaznamenány textové hodnoty objektu pro pozdější snadnější orientaci. Podobně jsou upraveny i další třídy.

Sbírání dat z aktivit je zaznamenáváno odlišným způsobem. K zajištění zapsání dat je využita instance třídy `Application`, resp. jejího potomka `ApplicationClass`, v které je využita metoda `registerActivityLifecycleCallbacks`. Ta umožňuje implementovat rozhraní pro všechny fáze životního cyklu všech aktivit, které jsou v dané aplikaci použity. K účelu sbíraní dat o chování uživatele využijeme metody `onActivityResumed`, reprezentující fázi životního cyklu *resume*, a `onActivityPaused`, odpovídající fázi *pause*, ze kterých je možné zjistit čas, který uživatel strávil na dané aktivitě. Využita je také metoda `onActivityCreated`, která slouží k vytvoření nové *session*, pokud se jedná o první spuštěnou aktivitu v rámci aplikace.

Mezi speciální data, která jsou sbírána patří informace o pádu aplikace. Pokud nastane situace, že uživateli při používání dojde k pádu aplikace, pak je zjištěna výjimka, která tento pád způsobila a data s ní související. Všechna tato data zajistí třída `ExceptionHandler` a uloží do lokální databáze. Později mohou být využita k vydání opravných aktualizací.

Kromě dat, která jsou sbírána automaticky pomocí výše popsaných možností, lze také zaznamenávat různé informace v libovolných částech kódu aplikace. K tomu slouží třída `LogEvent` a její statická metoda `event`, která umožňuje zadat proměnný počet parametrů typu `String`. Tato funkce bude využívána pro sledování vlastních událostí.

3.2.3 Ukládání dat

Data sbíraná komponentami je nutné ukládat do lokální databáze. Aby ukládání dat nenarušilo běh aplikace, je nutné, aby byly ukládány na pozadí v jiném než hlavním vlákně. Pro tento způsob práce s daty připadaly v úvahu dvě možnosti - využít třídy `AsyncTask` nebo `Service`.

Jednou z možností pro zápis dat pomocí třídy `AsyncTask` by bylo možné vytvářet instanci třídy s každým záznamem, který má být vložen do databáze. V tomto případě by bylo nutné zajistit zápis dat ve správném pořadí a čekat až bude konkrétní instance třídy `AsyncTask` ukončena, aby mohla být spuštěna nová. Toto řešení není vhodné zejména z paměťových a časových důvodů. Druhá možnost využití třídy `AsyncTask` – je vytvořena a dlouhodobě spuštěna s aplikací jediná instance této třídy a během jejího běhu jsou postupně zpracovávány požadavky pro zápis do databáze. Třídu `AsyncTask` však dle dokumentace není vhodné používat na delší úlohy než je několik vteřin, a proto ani tato možnost nebyla použita.

Pro řešení zápisu do databáze je využita služba `Service`. Služba je vhodná pro dlouhodobé operace a může běžet v samostatném vlákně. V tomto případě postačuje pouze jednostranná komunikace s hlavním vláknem. Aby bylo možné službu použít, je nutné ji zaznamenat do Manifestu knihovny. Při získání dat z aplikace jsou data zaslány službě, která se postará o jejich zapsání do databáze. Pro službu je specifické, že může být v jednu dobu vytvořena pouze jedna instance dané třídy. V případě tohoto řešení se jedná o službu `DatabaseService`, která je právě potomkem třídy `Service`.

Komunikace probíhá pomocí instance třídy `Intent` a volání statické metody `call` třídy `DatabaseService`. Ta zajišťuje případné spuštění služby a přeposlání dat službě pomocí instance třídy `Operation`. Služba využívá statického seznamu objektu třídy `Operation` a vkládá do něj jednotlivá data, která mají být uložena do databáze. Součástí služby je instance třídy `Runnable`, která vezme vždy první prvek seznamu a provede zápis do databáze. Tento proces je opakován dokud není seznam prázdný. Pokud je seznam prázdný, pak služba ověřuje prázdnost seznamu v zadaném intervalu 3 sekund. Pokud se po dobu 30 sekund nevloží do seznamu položka k zápisu do databáze, předpokládá se, že uživatel aplikaci nevyužívá a služba je ukončena. Aktivována je poté až v době, kdy je nutné opět zapsat další data. Vypínáním služby je šetřena baterie zařízení.

3.2.4 Odesílání dat

Data ukládaná v databázi zařízení je nutné odesílat na server, aby se nehromadily v zařízení a bylo je možné použít při jejich analýze. Aby odesílání dat nijak neomezovalo aplikaci, byla k této činnosti vytvořena služba `SyncService`, která je potomkem třídy `IntentService`. Využití této třídy je výhodné pro jednorázové operace, u kterých proběhna potřebná část kódu a po jeho dokončení se služba sama ukončí. Služba pro synchronizaci dat na server je

spuštěna vždy pod podmínkou, že zařízení je buď připojeno k WiFi, nebo překročilo délku zvoleného intervalu od poslední synchronizace – ten byl nastaven na 1 den.

O zasílání dat se stará třída `ServerCommunication`. Nejprve získá všechny *session* z databáze, které již byly ukončené a s nimi spojená data. Tato data jsou reprezentovány modelovými třídami `Session`, `Crash` a `Data`. Informace jsou pak pomocí třídy `JSONConverter` převedeny do formátu JSON. Takto vytvořený řetězec je poté zaslán metodou POST v těle HTTP / HTTPS požadavku na server, kde je přijat a zpracován. Pokud vše na serveru proběhně v pořádku, pak je serverem odpovězeno příznakem, který značí úspěšné uložení do centrální databáze a možnost smazat odeslaná data v lokální databázi. V případě, že se zápis na serveru nepovede, dojde k pokusu o zaslání dat později.

K odesílání dat na server dochází ve dvou případech. Prvním případem je spuštění samotné aplikace, při které je současně spuštěna samotná služba pro synchronizaci. V druhém případě je služba inicializována změnou připojení k síti. Pokud bylo zařízení připojeno k internetu přes mobilní data a připojilo se na WiFi, pak je také spuštěna synchronizace. Tento způsob je realizován pomocí třídy `NetworkChangeReceiver`. Ten musí být také patřičně zaznamenán v Manifestu knihovny.

3.2.5 Oprávnění

OS Android od verze 6.0 Marshmallow změnil svou politiku schvalování oprávnění a od této verze vyžaduje některá oprávnění schvalovat uživatelem přímo v aplikaci. Knihovna využívá některá oprávnění pro svou činnost. Jsou jimi připojení k internetu nebo sledování stavu připojení k síti. Tento typ oprávnění však není nutné uživatelem potvrzovat, ale jsou schváleny systémem automaticky. Knihovna dále využívá zápisu a čtení z úložiště zařízení, což už je typ oprávnění, které je potřeba potvrdit uživatelem. Z tohoto důvodu je nutné, aby si knihovna vyžádala potvrzení uživatele. Žádost o oprávnění je uživateli zobrazena v obvyklém systémovém dialogu, ihned při prvním spuštění aplikace. Toto je tak jediný viditelný zásah knihovny do aplikace, který je ovšem pro správnou funkci knihovny nezbytný.

Kapitola 4

Desktopová aplikace

V této kapitole je popsán celkový koncept aplikace a postup při návrhu této části. Součástí jsou také podkapitoly, které se zabývají prací s daty a komunikaci se serverovou částí. Druhá část kapitoly je zaměřena na implementaci této aplikace. V závěru jsou také zmíněny knihovny, které byly využity v rámci implementace.

Vizuální část systému bude zpracována v podobě desktopové aplikace a bude obsahovat dvě části. Hlavní část aplikace bude zobrazovat informace týkající se práce s mobilní aplikací. K tomuto zobrazení bude nejčastěji využívat různé typy grafů a tabulky. Druhou částí bude modul pro import knihovny a samotná knihovna. Tento modul umožní uživateli systému projít průvodcem pro import knihovny do mobilní aplikace a upraví dle potřeby původní zdrojové kódy aplikace. Veškerá data potřebná k práci s klientskou aplikací jsou uložena v databázi na serveru, tudíž je nutné připojení k internetu.

4.1 Návrh aplikace

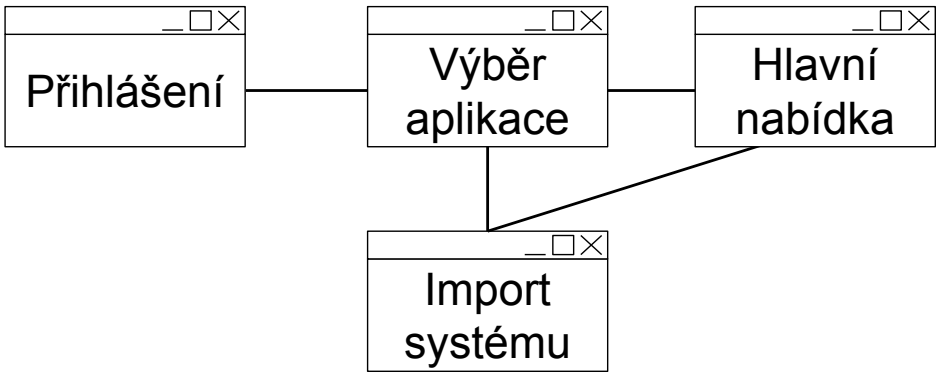
Tato část popisuje návrh desktopové aplikace. V jednotlivých podkapitolách je rozebrána struktura aplikace, její práce s daty, která jsou uložena na serveru a funkcionální návrh modulu pro import knihovny i samotné klientské aplikace pro grafické zobrazení dat.

4.1.1 Struktura aplikace

Celá aplikace bude složena z několika obrazovek, mezi kterými se může uživatel pohybovat. Schéma struktury aplikace, kde jsou rozvrženy jednotlivé obrazovky a přechody mezi nimi, je zobrazeno na obrázku 4.1. Při spuštění aplikace bude zobrazena přihlašovací obrazovka, která bude požadovat zadání serveru, přihlašovacího jméno a odpovídající heslo. Po úspěšném přihlášení se uživatel dostane na obrazovku, kde si vybere aplikaci, s kterou chce pracovat. V případě, že chce vytvořit v systému novou aplikaci, pak přejde na obrazovku modulu pro import systému do mobilní aplikace. Pokud uživatel vybral existující aplikaci, pak se přesune na hlavní obrazovku, kde si může vybrat z funkcí nabízených aplikací. Z této obrazovky lze také vstoupit do nastavení nebo modulu pro import a provést aktualizaci odpovídající aplikace.

4.1.2 Práce s daty a komunikace

Veškerá data, která desktopová aplikace využívá, jsou umístěna v databázi na serveru. Pokud chce aplikace nějaká data využívat, je nutné k tomu využít komunikační rozhraní



Obrázek 4.1: Schéma struktury obrazovek desktopové aplikace

na straně serveru. Přenos dat je vždy inicializován desktopovou aplikací. Komunikace je založena na zasílání HTTP / HTTPS požadavku ze strany klienta a odpovědí serveru. Komunikace probíhá v případě ověřování přístupových údajů při přihlášení do aplikace, získávání obsahu jednotlivých dashboardů, používání konkrétní funkce nebo při komunikaci modulu pro import se serverem. Struktury dat jsou přenášeny ve formátu JSON. Ukládání dat do databáze a její struktura jsou blíže popsány v kapitole 5. Na klientské aplikaci již žádné operace se sledovanými daty neprobíhají. V případě použití funkce jsou zaslány informace, o jakou funkci jde a s jakými parametry má být vyhodnocena. Server tento požadavek zpracuje a v odpovědi vrací pouze výsledná data, která jsou přenesena do grafické podoby.

4.1.3 Výběr platformy

Pro implementaci desktopové aplikace byla zvolena platforma JavaFX. Tato platforma, založená na Javě, slouží pro vývoj desktopových a RIA[1] aplikací. Tato platforma postupně nahrazuje Swing jako standardní knihovnu pro Java SE. JavaFX přináší podporu multimédií, animací, grafů, CSS a samotná obsahuje velké množství komponent. Tato platforma klade důraz zejména na jednoduchost tvorby a dává tak vývojáři možnost soustředit se pouze na podstatnou část práce. Výhodou této platformy je případný jednoduchý možný přechod na mobilní nebo webovou aplikaci. Pro studium základů platformy JavaFX a během implementace byly čerpány informace z knihy [10].

4.1.4 Model-View-Controller

Architektonický vzor Model-View-Controller (MVC) je softwarová architektura, která rozděluje aplikaci na komponenty tří typů – jsou to právě Model, View a Controller. Každá z těchto komponent má na starost specifickou část aplikace. Základní myšlenkou tohoto vzoru je oddělení logiky od výstupu aplikace.

[1]RIA - Rich Internet Application - webová aplikace, která má grafické vlastnosti desktopové aplikace, většinou běžící ve webovém prohlížeči

Model

Model představuje strukturu, do které jsou ukládány data. Obsahuje také veškerou logiku zastřešující práci s těmito daty - nejčastěji načítání a výpis informací. Často je využíván současně s databází.

View

View představuje vizualizaci dat z modelu – je to způsob, jakým jsou data zobrazena uživateli.

Controller

Controller je komponentou, která vytváří most mezi Modelem a View – je prostředníkem. Drží tak celý systém pohromadě a tyto komponenty propojuje. Kontroluje data, která jsou vkládána do modelu a v případě potřeby aktualizuje View. Samotný Model a View udržuje oddělené.

Použití

Návrhový vzor MVC je ve velké míře aplikován na tuto desktopovou aplikaci. Aplikace nevyužívá žádnou lokální databázi, proto je nutné všechna data, která aplikace využívá, stahovat ze serveru. Model umožňuje tato data ukládat do struktur a dále s nimi pracovat. Vzhledy jednotlivých obrazovek jsou zpracovány v FXML souborech. Tyto soubory obsahují značky reprezentující jednotlivé grafické komponenty a jejich uspořádání. Framework JavaFX zpracovává tyto soubory a vytváří konkrétní grafickou podobu obrazovek. Takto vytvořené grafické uspořádání komponent představuje View. Framework JavaFX umožňuje spojení View s Controllerem pomocí speciálního atributu v FXML souboru. Toto spojení umožní jednoduché mapování grafických komponent z FXML souboru do jeho Controlleru a jejich snadnější obsluhu. Pro každý View tak existuje právě jeden Controller, který v něm zajišťuje změny. Controller také zpracovává události, které pochází od uživatele a reaguje na ně odpovídajícím způsobem. Zajišťuje také spojení se serverem – odesílání a přijímání dat a komunikaci s modelem.

4.1.5 Modul pro import knihovny

Modul pro import knihovny je část aplikace, která zajišťuje proces vložení systému do mobilní aplikace. Lze jej použít buď na zcela novou mobilní aplikaci, nebo aplikaci, která systém pro sběr dat již obsahuje, ale prošla aktualizací. Android Studio je oficiálním vývojovým prostředím pro vývoj mobilních aplikací na OS Android. Modul při své činnosti využívá znalosti adresářové struktury jeho poslední verze[2], je proto nutné, aby projekt měl stejnou adresářovou strukturu, kterou poskytuje právě tato verze. Vývojové prostředí využívá automatizační nástroj Gradle pro vytvoření instalačního balíčku. Průvodce importem probíhá v několika krocích, kde si uživatel (vývojář) aplikace nastaví nabízené možnosti sledování dle svých potřeb.

[2]K datu 27. 12. 2016 je aktuální stabilní verze Android Studio 2.2.3

Kroky průvodce importem knihovny do mobilní aplikace

1. Uživatel zvolí adresářovou cestu vedoucí k projektu vytvořeném pomocí Android Studia podporované verze. Modul ověří strukturu projektu a vyhledá důležité informace z konfiguračních souborů nástroje Gradle. Aby byly zachovány zdrojové soubory aplikace a nedošlo k jejich přepsání, je také nutné zadat cílový adresář. Pokud se jedná o novou aplikaci, je požadován název projektu, pod kterým bude aplikace uložena v systému.

2. Každá konkrétní verze mobilní aplikace, která je používána, poskytuje specifickou funkcionalitu. Popis funkcionality, případně změn oproti předchozí verzi by měl být zapsán během importu do příslušného pole. Tyto informace by měly později sloužit pro lepší přehlednost funkcí v různých verzích aplikace.

3. Samotná knihovna poskytuje pouze sběr obecných dat. Pokud chce uživatel sledovat konkrétní činnost uživatele, například využití specifické funkce aplikace, pak tento modul poskytuje možnost vložit do zdrojových kódů sledování vlastní události. Příslušná akce je pak zaznamenána pomocí vhodně zvoleného identifikátoru a případně dalších parametrů.

4. Po dokončení nastavení individuálních sledování modul ukončí průvodce importem a začíná samotný proces importování do zdrojových kódů aplikace. Tento proces bude podrobněji rozebrán v kapitole 4.2.

Pro sestavení instalačního balíčku a následnou distribuci je vhodné využít vývojové prostředí nebo nástroj Gradle.

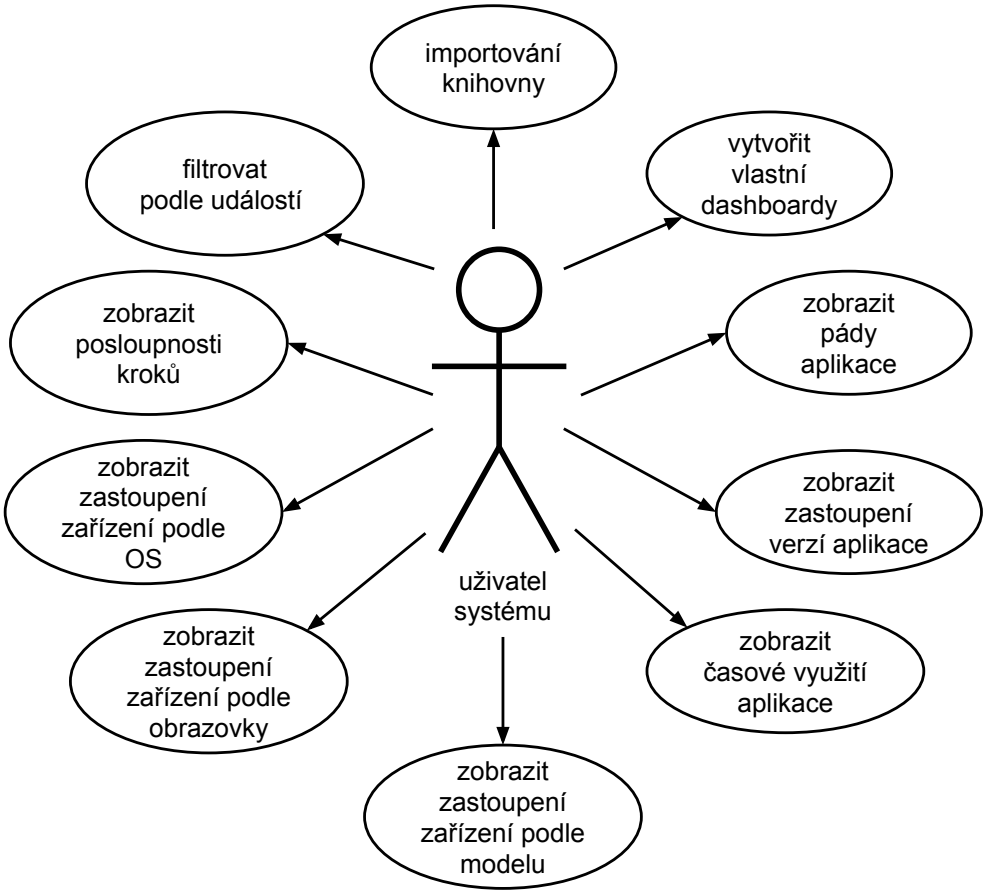
4.1.6 Klientská aplikace

Klientská aplikace bude nabízet spoustu možností zobrazení informací o používání aplikace. Případy užití pro uživatele klientské aplikace jsou uvedeny na obrázku 4.2 včetně možnosti importování knihovny pomocí modulu pro import. Jednotlivé funkce budou snadno parametrizovatelné tak, aby uživateli stačilo vybrat z nabídky poskytnuté systémem a nemusel zdlouhavě zadávat ručně vstupní parametry. Většina funkcí bude povinně časově parametrizována, aby lépe zobrazovali sledované data. Daný trend lze sledovat lépe v časovém úseku posledního týdne nebo měsíce, než dlouhodobě za několik let. Výjimku tvoří funkce, které pracují se statickými daty mobilních zařízení jako je velikost displeje nebo nebo značky telefonu.

Dashboard

Nejdůležitější částí desktopové aplikace bude přehledné zobrazení dat. Každého uživatele systému budou u určité aplikaci zajímat jiné informace. Avšak určitý uživatel bude pravděpodobně chtít sledovat dlouhodobě stejné informace. Aplikace tedy bude nabízet možnost vytvoření vlastních dashboardů[3]. Uživatel tak může vytvořit několik grafů či tabulek a logicky umístit do jednotlivých dashboardů k sobě ty, které spolu souvisí. Uživatel si tak přizpůsobí rozmístění grafů svým potřebám.

[3]Dashboard - místo, kde uživatel může vložit často využívané grafy a později k nim rychleji přistupovat



Obrázek 4.2: Diagram případů užití

Grafy

Pro zobrazení a přehlednost budou využity zejména jednoduché náhledy v podobě sloupcových, spojnicových nebo výsečových grafů. V případě potřeby budou doplněny legendou, tabulkou, či textovým popiskem. Grafy budou interaktivní – při najetí kurzoru nad konkrétní oblast či uzel grafu budou zobrazeny detailní textové informace.

4.1.7 Technologie

Tato kapitola popisuje technologie použité během vývoje celé desktopové aplikace. První část kapitoly je věnována vývojovému prostředí, které bylo využito během implementace systému. Závěr kapitoly je zaměřen na externí knihovny použité v desktopové aplikaci a jejich konkrétní využití.

Vývojové prostředí

Pro aplikaci byla navržena platforma JavaFX a byla tak implementována v programovacím jazyce Java. Proto je nutností instalace Java Development Kitu (JDK), který obsahuje soubor základních nástrojů pro vývoj aplikací v tomto jazyce. Pro snadnější vývoj, testování a ladění aplikace je vhodné využít vývojové prostředí, které může usnadnit práci na projektu a zajistit integraci všech potřebných nástrojů.

Pro vývoj desktopové aplikace bylo možno vybrat z několika vývojových prostředí (IDE). V současné době je možné pro vývoj v JavaFX využít například IntelliJ IDEA, NetBeans IDE nebo Eclipse. Pro vývoj nakonec bylo zvoleno IntelliJ IDEA, dostupné z webu [8]. Důvodem byla především osobní zkušenost s výše zmíněnými IDE. Navíc v IntelliJ IDEA nebyla nutná žádná dodatečná instalace modulů na rozdíl od jiných prostředí. Pro implementaci tohoto systému byla využita Java 8. Z této nejnovější verze jazyka byla využita zejména možnost pracovat s lambda výrazy v případech, kdy je implementováno rozhraní.

Volitelným nástrojem pro vytváření grafického rozhraní je Java Scene Builder. Tento grafický designer, pocházející od firmy Oracle, není přímo obsažen v žádném z vývojových prostředí, ale je samostatně fungující aplikací, kterou je možné integrovat právě do IntelliJ IDEA.

Knihovny

Pro správné fungování desktopové aplikace budou využity dvě externí knihovny. V této části budou popsány jejich možnosti a využití.

json-simple[4] – tato knihovna poskytuje možnost práce s řetězci ve formátu JSON. Pomocí této knihovny je možné sestavovat i parsovat řetězce ve formátu JSON. Tato funkcionalita bude využita při sestavování dotazu na server, kdy je potřeba vytvořit požadavek v tomto formátu a také při parsování po obdržení odpovědi ze serveru. Knihovna je distribuována pod licencí Apache License 2.0.

commons-io[5] – tato knihovna poskytuje mnoho tříd pro ulehčení práce se soubory a adresáři. V systému bude využita pouze během importu, kde bude potřeba kopírovat soubory a adresáře. Knihovna je distribuována pod licencí Apache License 2.0.

4.2 Modul pro import

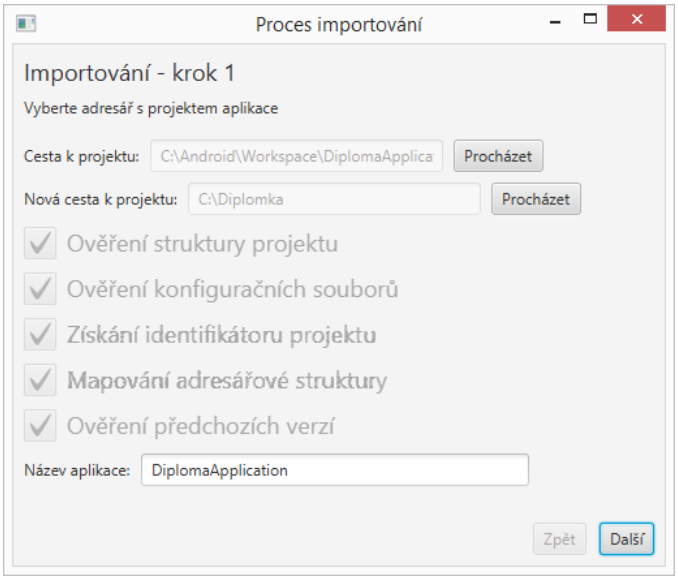
V této části je popsána implementace části, která zajišťuje import systému do mobilní aplikace. Tento průvodce je rozdělen do několika kroků, které na sebe plynule navazují a budou popsány v samostatných sekcích. Během jednotlivých kroků jsou získávány potřebné informace, které se postupně posílají do kroků následujících. Posledním krokem je proces importování, který všechny informace zpracovává a vykoná potřebné operace. Častou operací, která je prováděna během importu, je získávání informací ze souborů, ať už se jedná o konfigurační soubory nebo zdrojové soubory ve formátu XML nebo Java. K vyhledávání textových údajů v souborech jsou použity regulární výrazy s využitím tříd `Pattern` a `Matcher` z vestavěné knihovny `java.util.regex`.

4.2.1 Průvodce importem - krok 1

V tomto kroku je vyžadováno zvolení adresáře, který obsahuje projekt a uživatel do něj chce importovat systém pro sledování. Může jít také o projekt, ve kterém je již sledovací systém obsažen, ale prošel aktualizací. Aby nedošlo k přepsání původních zdrojových kódů aplikace, je nutné, aby uživatel také zvolil cílový adresář, kde bude umístěna aplikace včetně importované knihovny. Pokud jsou tyto dva adresáře již zvoleny, pak je v několika krocích kontrolována struktura projektu a klíčových konfiguračních souborů. Tato obrazovka, která

[4] Dostupné z https://code.google.com/archive/p/json-simple/
[5] Dostupné z http://commons.apache.org/proper/commons-io/

představuje krok 1 při importování knihovny, je reprezentována třídou `Step1`. Její náhled je zobrazen na obrázku 4.3.

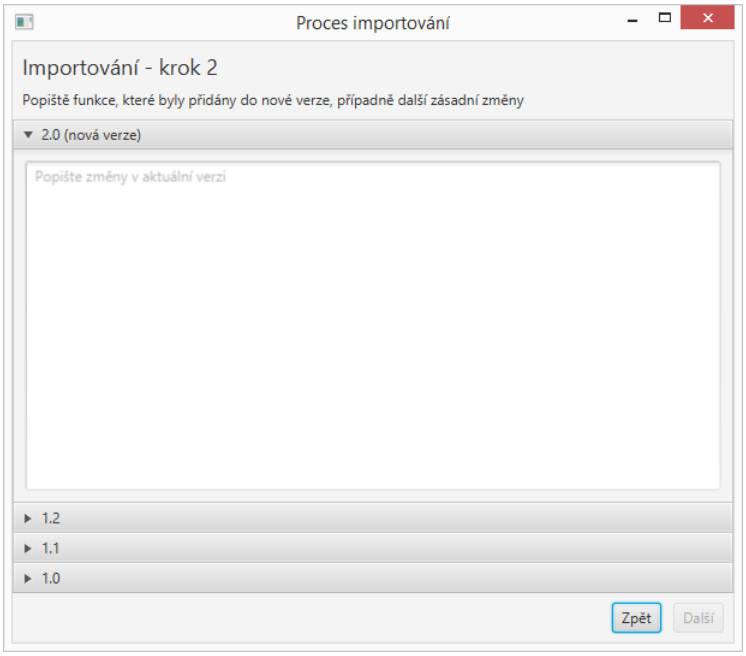


Obrázek 4.3: Průvodce importem - krok 1

Nejprve je ověřena struktura projektu a to pomocí metody `checkProject`. Systém zde kontroluje, zda zvolený adresář obsahuje konfigurační soubory systému Gradle pro vystavění aplikace. Poté jsou pomocí metody `findInfo` vyhledávány informace o projektu. Pro tuto metodu je klíčový soubor *settings.gradle*, který obsahuje seznam modulů aplikace. Jednotlivé moduly jsou pak umístěny ve složkách v hlavním adresáři aplikace. Každý modul musí obsahovat soubor *build.gradle*, který obsahuje informaci, zda se se jedná o hlavní modul aplikace nebo vedlejší. Podle toho je také určena struktura souboru. Z tohoto souboru hlavního modulu je zjištěn identifikátor aplikace a její verze. Předposledním krokem je namapování adresářové struktury aplikace, která bude využita v kroku 3. Tento proces je zajištěn pomocí rekurzivního volání metody `createBranch`, která postupně vytvoří model adresáře projektu. Posledním krokem je zjištění, zda aplikace již existuje v systému. V tomto případě je nutné ověřit tuto skutečnost skrze dotaz na databázi umístěnou na serveru. V případě, že se jedná jedná o novou aplikaci, která není v systému obsažena, pak je po uživateli požadováno jméno aplikace, pod kterou bude v systému uložena. V opačném případě jsou pouze staženy informace o předchozích verzích aplikace, které budou využity v kroku 2.

4.2.2 Průvodce importem - krok 2

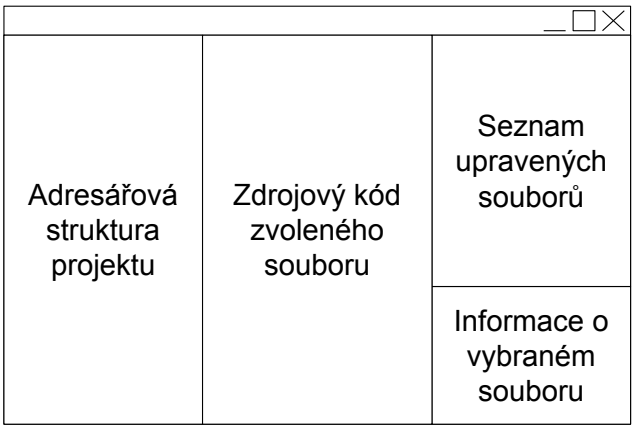
Na obrazovce odpovídající tomuto kroku je zobrazen přehled všech předchozích verzí mobilní aplikace, které jsou uloženy v databázi systému. Po uživateli je požadováno, aby doplnil odpovídající informace k aktuální verzi. Obsah tohoto textu je závislý na uživateli, avšak jeho primární použití by mělo sloužit k přehledu, jak byla vyvíjena aplikace a v jaké verzi aplikace byla doplněna konkrétní funkcionalita. Tato obrazovka odpovídá třídě `Step2` a její náhled je na obrázku 4.4.



Obrázek 4.4: Průvodce importem - krok 2

4.2.3 Průvodce importem - krok 3

V tomto kroku může uživatel sledovat a ručně editovat jednotlivé zdrojové soubory projektu. Její primární funkcí je vkládat volání statické metody knihovny - `LogEvent.event`, která umožňuje monitorování vlastních specifických dat do systému. Tato obrazovka je rozdělena do 3 částí. Obrazovka odpovídá kroku 3 a je reprezentována třídou `Step3` a její náhled je zobrazen na obrázku 4.5.



Obrázek 4.5: Průvodce importem - krok 3

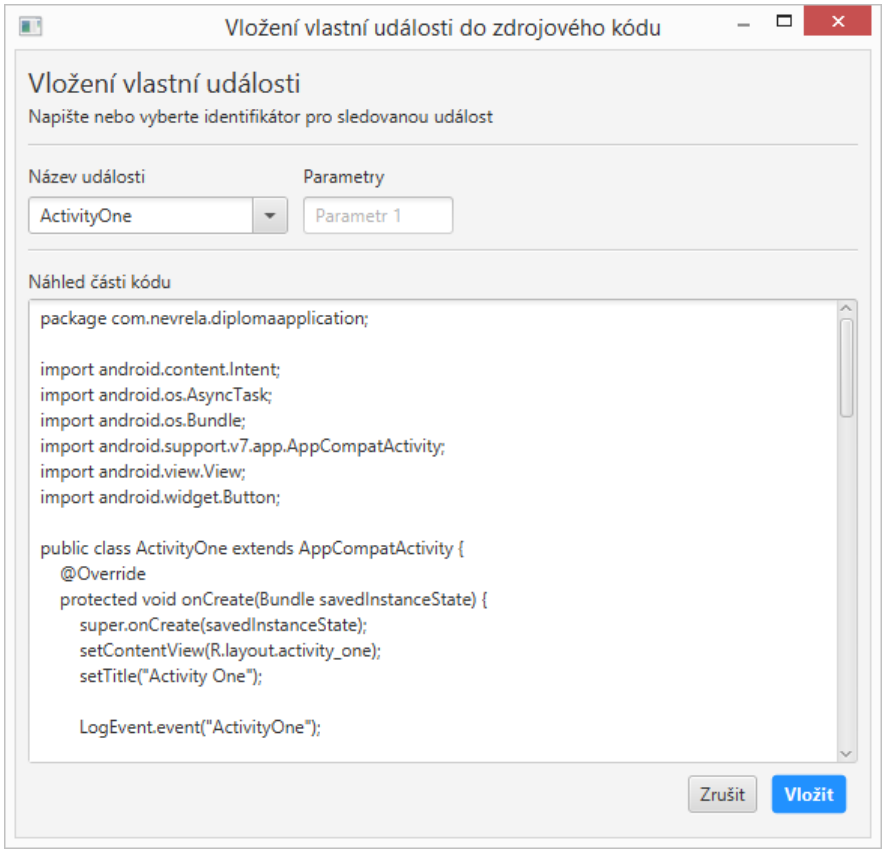
V levém sloupci je zobrazena adresářová struktura zdrojových kódů aplikace, která je reprezentována stromovou formou pomocí vestavěné komponenty `TreeView`. Jednotlivé položky stromové struktury, reprezentující složky nebo soubory, jsou implementovány pomocí třídy `TreeItemExpandable`. Tato třída, na rozdíl od `TreeItem`, ze které je odvozena, je

doplněna o funkcionalitu, která umožňuje řetězově rozbalit v sobě vnořené složky, pokud se jedná pouze o jedinou složku umístěnou v otevírané složce.

V prostředním sloupci je po zvolení souboru ve stromové struktuře zobrazen jeho obsah. Pokud chce uživatel ručně doplnit zdrojový kód, pak má možnost dopsat zde fragment kódu, který potřebuje. Pokud však chce uživatel vložit sledovanou události automatizovaně, zvolí soubor a místo v kódu, kde chce sledovanou událost vložit a stiskne tlačítko *Vložit událost*, které otevře dialog pro vložení vlastní události.

Pravý sloupec je složen ze dvou částí. Vrchní část obsahuje seznam souborů, které byly editovány v rámci tohoto importu. Kliknutím pravým tlačítkem na vybraný soubor nabídne uživateli smazání změn provedených v konkrétním souboru. Druhá část zobrazuje detailní informace o právě aktivním souboru – cestu k souboru, informaci, zda je soubor upraven, a vlastní události, které jsou sledovány v souboru. Sledované události jsou vyhledávány v textu pomocí regulárních výrazů.

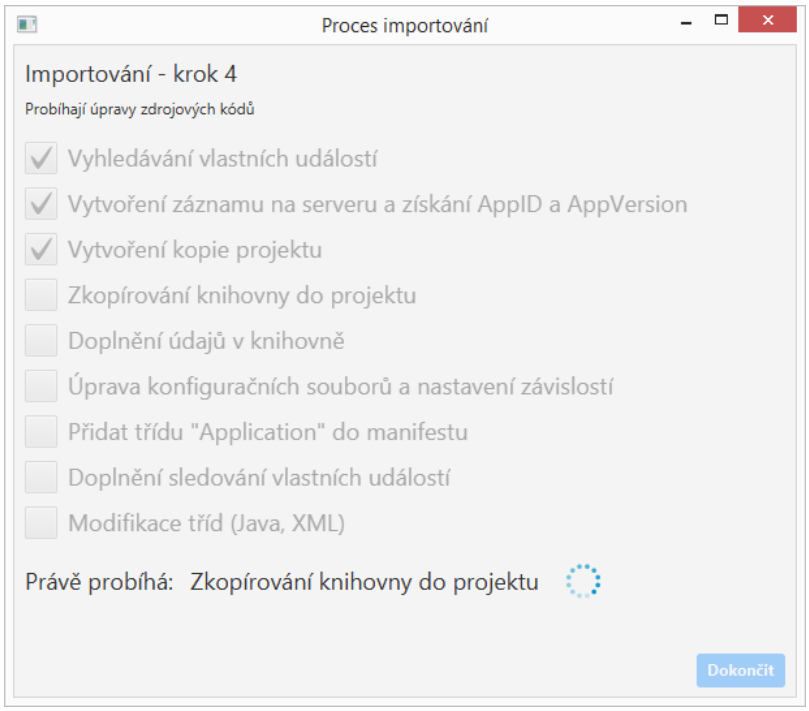
Dialog pro vložení vlastní události umožní pomocí nabídnutých textových polí vložit na požadované místo volání metody `LogEvent.event`. Všechny její parametry jsou datového typu `String` a jejich počet je volitelný, avšak musí obsahovat alespoň jeden parametr, který značí jméno sledované události. Při vyplňování jednotlivých parametrů metody automatizovaně přibývají další políčka pro možné další parametry. Po vyplnění je nutné potvrdit změny provedené v dialogu. Při této akci je v případě potřeby doplněn import pro třídu `LogEvent` do upravovaného souboru. Změny jsou ihned zobrazeny v předchozí obrazovce s obsahem souboru. Celý tento dialog je reprezentován třídou `DialogEventInsert` a jeho náhled je na obrázku 4.6.



Obrázek 4.6: Průvodce importem - vložení vlastní události

4.2.4 Průvodce importem - Proces importování

Tato část je posledním krokem průvodce pro import knihovny do aplikace. Jejím hlavním cílem je zpracovat požadavky, které byly nastaveny během všech kroků importu a vytvořit nový adresář s upravenými zdrojovými kódy. Tento proces je poměrně zdlouhavý, trvající od jednotek sekund do jednotek minut závisející na velikosti projektu, proto jsou tyto úpravy prováděny v samostatném vlákně implementovaném třídou `ImportThread`. Obrazovka odpovídající procesu importování pouze ukazuje aktuální stav v průběhu importování a je reprezentovaná třídou `Step4`. Její náhled je zobrazen na obrázku 4.7.



Obrázek 4.7: Průvodce importem - krok 4

Proces importování je složen z několika kroků:

1. Vyhledávání vlastních událostí.
   Jsou prohledány všechny soubory v jazyce Java obsažené v modulech. V každém souboru je pomocí regulárního výrazu vyhledávána statická metoda `LogEvent.event`. K tomuto je využita metoda `getOwnEvents`, která rekurzivně prochází adresářový model a ukládá názvy vlastních událostí obsažených ve zdrojových kódech aplikace.

2. Vytvoření nového záznamu na serveru.
   Aby bylo možné z mobilní aplikace sbírat data vázané k dané aplikaci a její verzi, je nutné získat identifikátor nové verze v databázi. Proto nejprve musí být zaslán požadavek pro vytvoření záznamu v databázi na serveru. V odpovědi pak získáme potřebné údaje.

3. Vytvoření kopie projektu.
   Aby nebyly přepsány původní zdrojové kódy, je vytvořena kopie projektu pomocí metody `copyProject`, která využívá knihovnu *commons-io*.

4. Zkopírování knihovny do projektu.
   Knihovna, která je importována do zdrojových kódů aplikace, je zabalena ve formátu ZIP. Nejprve musí být zkopírována do nového adresáře a poté rozbalena pomocí metody `copyLibrary`. Zbylý soubor ve formátu ZIP je smazán. Opět je využita knihovna *commons-io*.

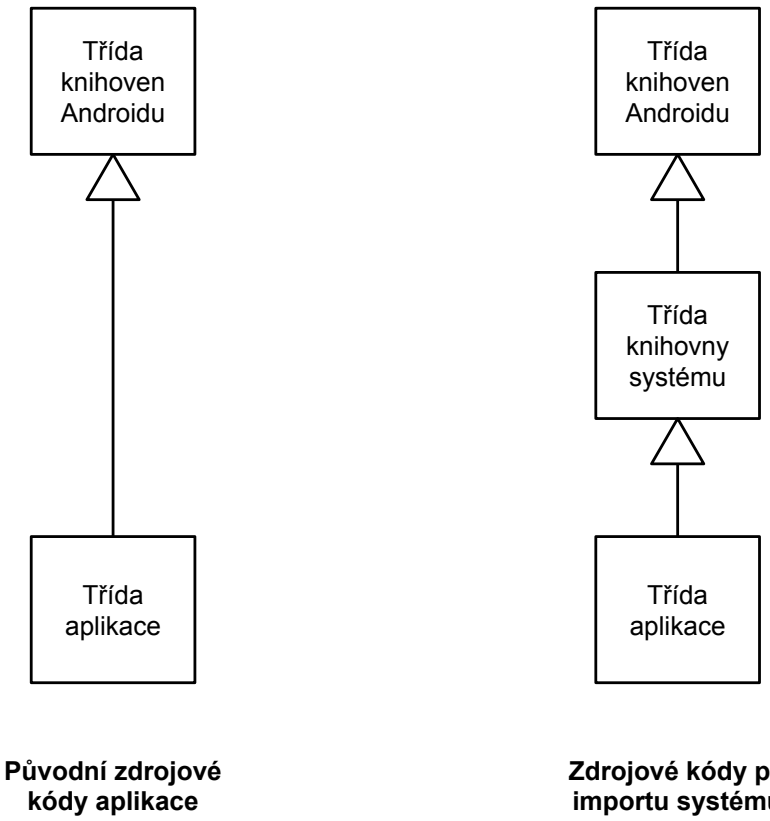
5. Doplnění údajů v knihovně.
   K uložení dat získaných z databáze, podle kterých je daná aplikace a její verze identifikována v rámci systému jsou uloženy do třídy `ApplicationClass`, která je potomkem třídy `Application`. Objekt této třídy představuje zároveň instanci celé aplikace, takže v rámci běhu aplikace je vytvořena právě jednou. Součástí třídy `ApplicationClass` jsou třídní proměnné, které jsou nastaveny podle hodnot získaných ze serveru. Tyto hodnoty jsou poté dostupné celé aplikaci v rámci aplikačního kontextu.

6. Úprava konfiguračních souborů a nastavení závislostí.
   Nejprve je doplněn modul do konfiguračního souboru *build.gradle* celé aplikace. Poté jsou prohledány soubory *build.gradle* všech modulů projektu a případně doplněny závislosti na tento nový modul. Tuto část zaštiťuje metoda `changeConfFiles`. Poté mohou být používány třídy knihovny v rámci ostatních modulů.

7. Přidání třídy ApplicationClass do manifestu.
   Aby byly proměnné dostupné v rámci aplikace, je nutné, aby třída `ApplicationClass`, byla součástí třídy, která je zvolena v manifestu jako třída odpovídající celé aplikaci. V tomto případu nastává několik možností, které mohou nastat. Tuto třídu není v aplikaci povinné definovat. V tomto případě je pouze do manifestu doplněna samotná třída. Pokud se jedná o aktualizaci aplikace a v manifestu je již tato třída doplněna, není nutné do souboru manifestu nijak zasahovat. Poslední možností je nastavená třída programátora, která je přímým či nepřímým potomkem třídy `Application` z knihovny OS Android. V tomto případě je nutné nalézt třídu v rodičovské hierarchii, která je přímým potomkem třídy `Application` a přepsat jejího rodiče na třídu `ApplicationClass`. Je tak vložen mezičlánek mezi třídu implementovanou programátorem a třídu, která je součástí OS Android.

8. Doplnění změněných souborů.
   Soubory, které byly změněny, jsou nalezeny v novém adresáři pomocí relativní cesty. Jejich kódy jsou v rámci metody `completeChangedFiles` přepsány těmi, které byly zaznamenány v rámci kroku 3.

9. Modifikace tříd Java a XML.
   Zdrojové soubory v jazycích Java i XML jsou procházeny a v nich vyhledávány objekty tříd, které byly použity k vytvoření tříd v knihovně. Třídy těchto objektů jsou nahrazeny třídami použitými v knihovně. Tímto je zajištěno sbírání dat v běžně používaných třídách. Náhled schématu dědičnosti mezi jednotlivými třídami původní aplikace, knihovny systému a knihoven Androidu je zobrazen na obrázku 4.8. Tento proces zpracovává třída `Subs`, která obsahuje seznam použitých komponent a tříd, které je mají nahradit.



Obrázek 4.8: Schéma dědičnosti tříd před a po importu sledovacího systému

4.3 Klientská aplikace

Tato kapitola popisuje implementaci klientské aplikace. Součástí této kapitoly bude popis jednotlivých obrazovek, jejich náhled a funkce, které nabízejí. Hlavní účel klientské aplikace je zpřístupnění agregovaných dat v takové podobě, aby byla uživateli systému užitečná.

4.3.1 Přihlašování

První obrazovkou je přihlašovací formulář, který umožní uživateli přihlášení do systému. Uživatel vyplní přihlašovací jméno, heslo a adresu serveru, na kterém je umístěno rozhraní systému a databáze. Každý uživatel systému má svůj osobní účet a po přihlášení jsou k jeho účtu staženy osobní data. Zejména se jedná o data spojené s dashboardem. Při přihlášení je také pomocí metody `downloadApps` stažen přehled aplikací, které jsou obsažené v databázi. Obrazovka přihlášení je reprezentována třídou `Login`.

4.3.2 Výběr aplikace

Obrazovka výběru aplikace následuje ihned po přihlášení do systému. Uživateli jsou nabídnuty aplikace uložené v systému, s kterými může pracovat. U každé aplikace je možné sledovat, jaká je aktuální verze aplikace uložená v systému a kdy byla nahrána. Po zvolení aplikace se uživatel přesune na hlavní obrazovku aplikace, kde již může vybírat mezi nabízenými funkcemi systému. Je také možné přidat do systému novou aplikaci a přejít

tak na obrazovku pro import knihovny do mobilní aplikace. Tato obrazovka odpovídá třídě `AppChooser` a její náhled je na obrázku 4.9.

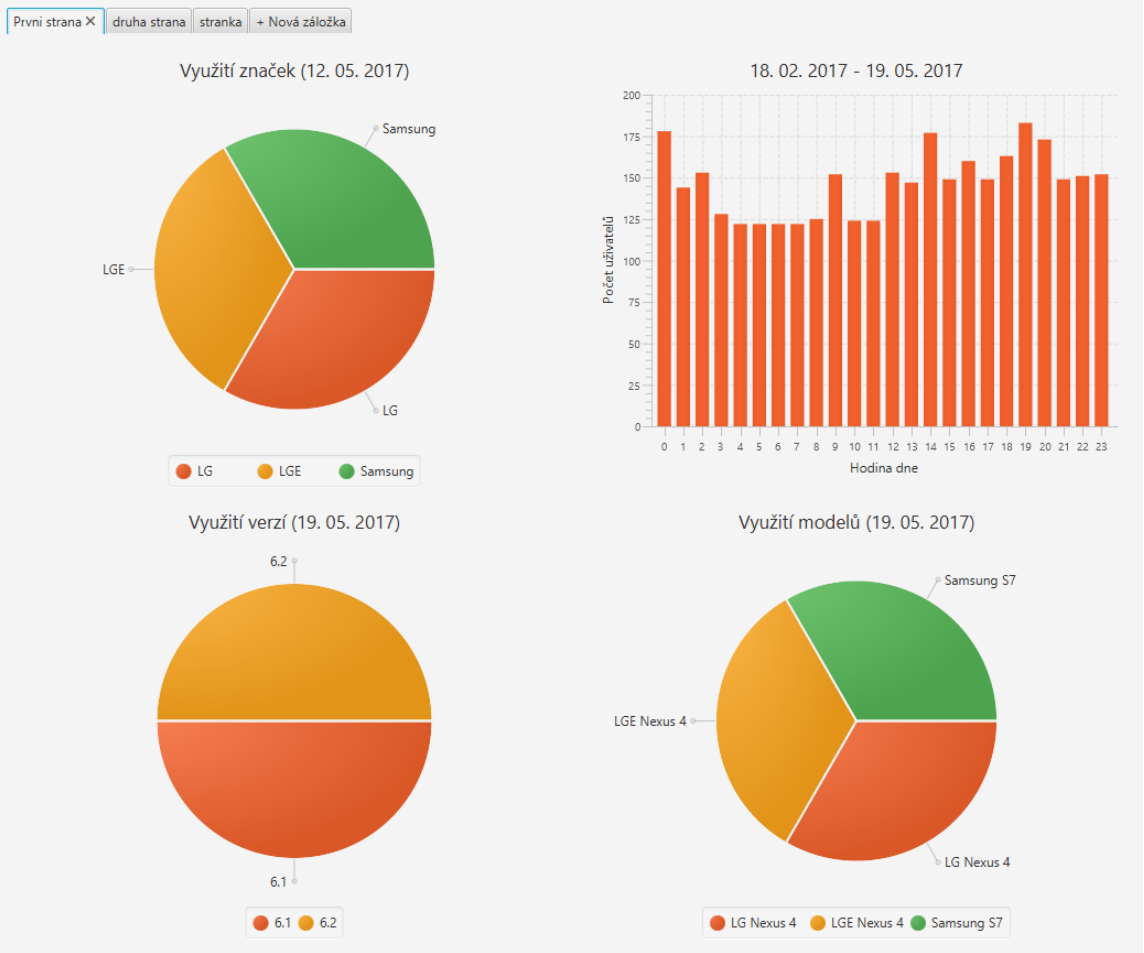
Obrázek 4.9: Obrazovka výběru aplikace

4.3.3 Hlavní obrazovka

Tato obrazovka již nabízí uživateli všechny funkce a možnosti, kterými systém disponuje. Horní část obrazovky je vyhrazena pro tlačítka administrace – přidání nové verze aplikace, výběr jiné aplikace nebo odhlášení ze systému. V levém sloupci obrazovky je zobrazen výčet funkcí, které lze aplikovat na data mobilní aplikace, kterou má uživatel momentálně zvolenou. V hlavním prostoru obrazovky jsou zobrazována data k aktuálně vybrané funkci. Při prvním vstupu na hlavní obrazovku je zde zobrazen dashboard, který je personalizován dle přihlášeného uživatele.

4.3.4 Dashboard

Obrazovka dashboardu umožňuje uživateli zobrazení předem vybraných grafů, které jsou aktuální k současnému datu. Tato funkce by měla umožnit snadnější přístup k funkcím, které jsou využívány opakovaně. Dashboard je složen ze záložek, které obsahují grafy. Pokud chce uživatel přidat graf do dashboardu, pak tak učiní u konkrétní funkce pomocí ikony hvězdičky. Tato funkce umožňuje přidávat a odebírat grafy na jednotlivých záložkách a také vytvářet a mazat samotné záložky. V každé záložce je několik menších grafů. Pokud si chce uživatel podrobně zobrazit daný graf, je možnost každý graf z dashboardu zvětšit. Ukázka jedné záložky z dashboardu je na obrázku 4.10.



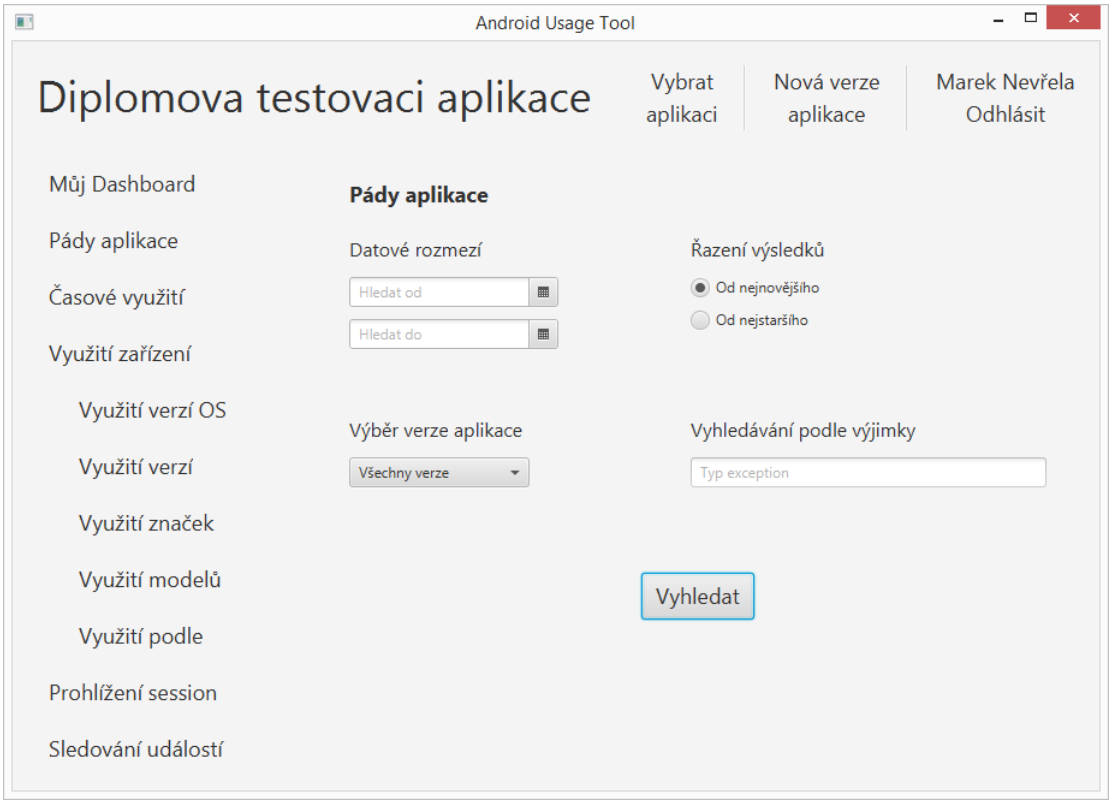
Obrázek 4.10: Ukázka záložky dashboardu

4.3.5 Pády aplikace

Knihovna importovaná do mobilní aplikace je schopna odchytávat výjimky (Exception) v rámci celé aplikace. Předtím, než je aplikace ukončena systémem, jsou zapsány informace o aktuální *session* a informace o odchycené výjimce do lokální databáze aplikace. Důležitou položkou je příčina, která danou výjimku vyvolala. Tato funkce by měla sloužit především k odladňování aplikace a hledání chyb. Cílem tak je minimalizovat počet chyb, které budou v další možné aktualizaci aplikace.

Jednotlivé záznamy o pádech aplikace lze filtrovat podle několika kritérií. Uživatel může zvolit období, ve kterém chce vyhledávat pády aplikace a pořadí, v jakém budou řazeny. Dále může zvolit, zda chce vyhledávat ve všech verzích aplikace nebo pouze v konkrétní. Případně může zvolit konkrétní název výjimky, kterou hledá. Obrazovku volby parametrů je na obrázku 4.11.

Všechny tyto parametry jsou zpracovány do společného požadavku a odeslány na server. Server zasílá v odpovědi seznam záznamů, které odpovídají zadaným parametrům. Tyto záznamy jsou zobrazeny přehledně v tabulce jako na obrázku 4.12. Uživatel si může vybrat libovolný záznam a dvojklikem otevřít detailní pohled zvoleného záznamu. Příklad tohoto detailního pohledu můžete vidět na obrázku 4.13. Lze zde najít detailní informace o zařízení, *session* a příčiny vyvolání výjimky.



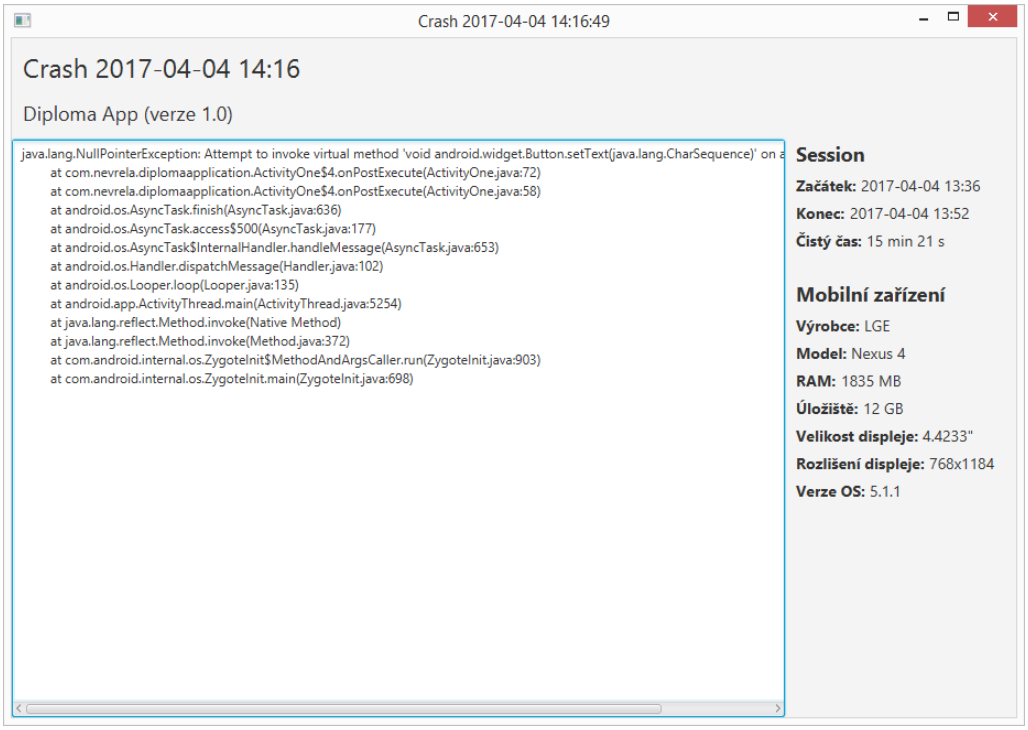
Obrázek 4.11: Volba parametrů pro funkci Pády aplikace

Výjimka	Chyba	Datum a čas
java.lang.NullPointerException	Attempt to invoke virtual method 'void andr...	2017-04-04 14:16:49
java.lang.NullPointerException	Attempt to invoke virtual method 'void andr...	2017-04-04 13:36:49
java.lang.NullPointerException	Attempt to invoke virtual method 'void andr...	2017-04-04 13:32:00
java.lang.NullPointerException	Attempt to invoke virtual method 'boolean ... Unable to start service com.nevrela.systemu...	2017-04-04 13:28:24
java.lang.NullPointerException	Attempt to invoke virtual method 'void andr...	2017-04-04 13:10:28
java.lang.NullPointerException	Attempt to invoke virtual method 'void andr...	2017-04-04 12:59:39
java.lang.NullPointerException	Attempt to invoke virtual method 'java.lang....	2017-04-04 12:48:19
java.lang.NullPointerException	Attempt to invoke virtual method 'java.lang....	2017-04-04 12:46:05
java.lang.NullPointerException	Attempt to invoke virtual method 'void andr...	2017-04-04 12:36:40
java.lang.NullPointerException	Attempt to invoke virtual method 'void andr...	2017-04-04 12:34:54
java.lang.NullPointerException	Attempt to invoke virtual method 'void andr...	2017-04-04 12:34:01
java.lang.NullPointerException	Attempt to invoke virtual method 'boolean ... Unable to start service com.nevrela.systemu...	2017-04-04 12:11:18
java.lang.NullPointerException	Attempt to invoke virtual method 'android.o... Unable to start service com.nevrela.systemu...	2017-04-04 12:00:12
java.lang.NullPointerException	Attempt to invoke virtual method 'void andr...	2017-04-04 11:58:14
java.lang.NullPointerException	Attempt to invoke virtual method 'java.lang....	2017-04-04 10:12:11
java.lang.NullPointerException	Attempt to invoke virtual method 'java.lang....	2017-04-04 10:11:08
java.lang.NullPointerException	Attempt to invoke virtual method 'java.lang....	2017-04-04 09:40:15

Obrázek 4.12: Přehled pádů aplikace

4.3.6 Časové využití

Funkce časového využití umožňuje uživateli systému sledovat, kdy je aplikace nejčastěji používána. Cílem tak je zjistit, jak často je aplikace využívána a v jakých částech dne.



Obrázek 4.13: Detailní pohled pádu aplikace

Z výsledků této funkce lze také vyčíst, v jaké fázi dne je aplikace nejméně využívána a tento údaj použít například k naplánování údržby serveru.

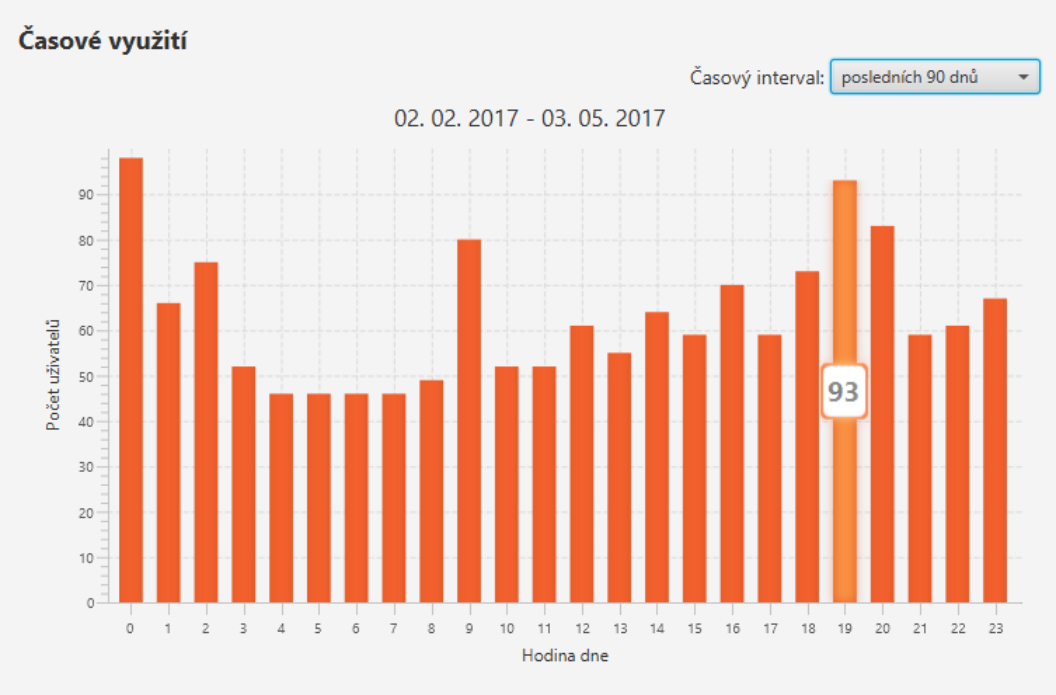
Výsledky této funkce jsou závislé pouze na jediném parametru. Tímto parametrem je interval v řádech dní, ze kterého jsou využita data pro výstup této funkce. Tento interval vždy končí současným datem a je možné vybrat z několika délek v řádech dní až po délku několika měsíců. V rámci používání aplikace by měl být trend dlouhodobějších statistik podobný. Pokud tomu tak není, může to indikovat změny v používání aplikace v rámci větších skupin uživatelů.

Pro zobrazení výstupů této funkce byl vybrán sloupcový graf, protože nejvhodněji reprezentuje výstupní data. Každý sloupec grafu představuje interval jedné hodiny. Na ose Y jsou pak zobrazeny počty uživatelů, kteří v určitou hodinu mobilní aplikaci používali. Příklad takového zobrazení lze vidět na obrázku 4.14. Tyto grafy jsou interaktivní, při najetí kurzorem je zobrazena přesná hodnota v daném sloupci tak, jak je zobrazeno na obrázku pro 19. hodinu.

4.3.7 Využití dat ze zařízení

Funkce využívající data ze zařízení umožňuje uživateli systému sledovat několik informací o zařízeních. Je možné zobrazit využití zařízení podle verze OS Android, značky nebo modelu zařízení a velikosti obrazovky. Kromě toho lze také prohlédnout zastoupení konkrétních verzí aplikace. Cílem je tak zjistit, jaké je složení cílové skupiny uživatelů a jak se tato skupina mění s přibývajícím časem.

Všechny tyto funkce jsou vázány k uživatelem zvolenému datu a vybrané aplikaci. Pro zobrazení výstupů těchto funkcí byl zvolen výsečový graf. Každá výseč reprezentuje určitou skupinu zařízení podle používané funkce. Tyto grafy jsou také interaktivní a po najetí



Obrázek 4.14: Funkce časového využití

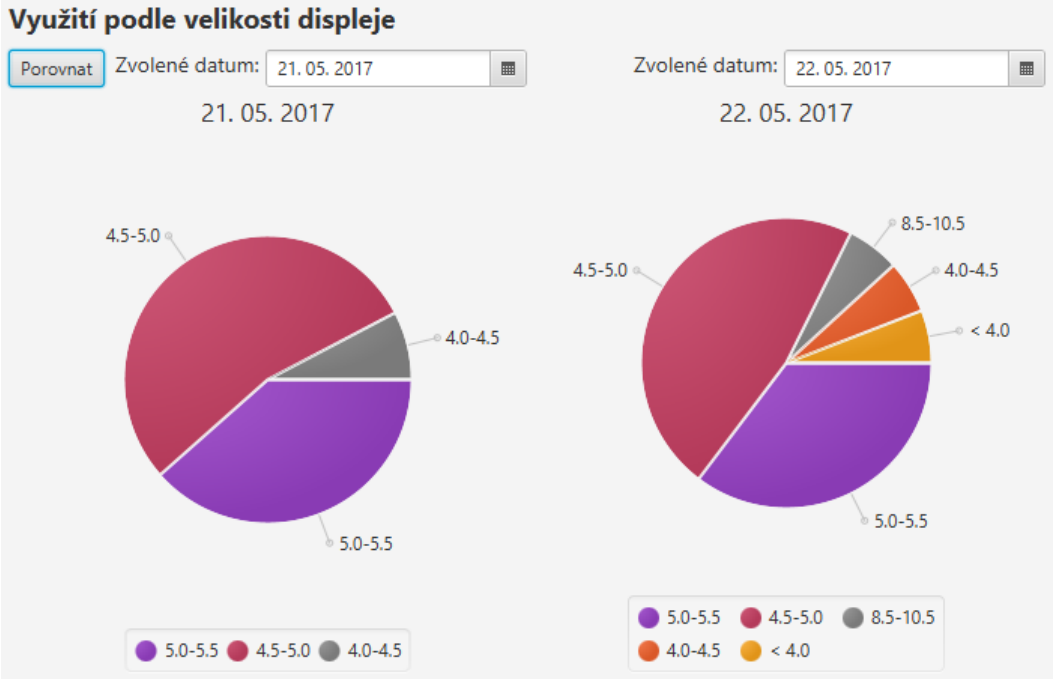
kurzorem nad výseč je zobrazena přesná hodnota, která odpovídá dané výseči. Pro možnost přehledného porovnání změn je navíc možné zobrazit vedle sebe dva výsečové grafy s různými daty.

Příklad grafu pro zobrazení dat podle velikosti obrazovky lze vidět na obrázku 4.15. U této funkce jsou velikosti obrazovek uspořádány do předem definovaných intervalů. Tyto intervaly byly zvoleny pro rozdělení skupiny uživatelů mobilních telefonů, a poté uživatele menších a větších rozměrů tabletů.

4.3.8 Prohlížení session

Funkce prohlížení session slouží k prohlížení konkrétních kroků uživatele ve vybrané *session*. Během používání mobilní aplikace jsou sbírána data – automaticky pomocí komponent knihovny nebo pomocí volání metody pro sběr vlastních událostí. Tato funkce umožní procházení všech záznamů, které byly k dané *session* vytvořeny krok po kroku. Možnost zobrazení jednotlivých kroků může být využita při testování aplikace – například při simulování chování uživatele v aplikaci. Pokud došlo k pádu aplikace, je možné vidět celý zaznamenaný běh aplikace, který tomuto pádu předcházel.

Úvodní obrazovka této funkce obsahuje tabulku, kde je obsažen přehled všech *session* za posledních 7 dní. Tento interval je možno zvolit dle nabídky, případně vybrat konkrétní den. Kromě volby dne umožňuje tato funkce zobrazit pouze *session*, které byly ukončeny pádem aplikace. Pro prohlížení kroků konkrétní *session* je potřeba dvojklikem vybrat záznam v tabulce. Při této akci je otevřeno nové okno aplikace s podrobnostmi o vybrané *session* včetně časových známek ke každé zaznamenané události.



Obrázek 4.15: Funkce využití podle velikosti obrazovky

4.3.9 Sledování událostí

Cílem této funkce je sledování událostí, které probíhají během *session*. Slouží k získání informace o tom, jak jsou sledované aktivity či funkce vytěžovány. Uživatel zvolí pomocí parametrů takové záznamy *session*, které budou považovány za základ výpočtu této funkce. Těmito parametry může být časové ohraničení zápisu *session*, minimální verze aplikace a obsah *session*. Obsah *session* může být zvolen na základě informace, která je obsažena v tabulce *Data*. Pokud jej tvoří vlastní událost, pak je využito klíčové slovo `ownEvent`, které je vyplňováno do políčka třída a její identifikátor do políčka akce. Parametry poté odpovídají parametrům ve sledované události. Ke správnému formátu zápisu nevlastních událostí napomůže tabulka sbíraných událostí 3.1. Pomocí této tabulky lze vyplnit třídu, zaznamenanou akci, či konkrétní hodnoty. Hodnot může být ke konkrétní akci více a v tomto případě odpovídají parametrům 1, 2 a 3, podle toho, která akce je zaznamenávána. Tímto jsou vybrány *session*, které tvoří základ výpočtu.

Druhou částí je vybrání události nebo skupiny událostí, které mají být sledovány. Podobným způsobem jako při vytváření základu funkce jsou zadávány parametry pro filtrování informací obsažených v tabulce *Data*. Takto nastavené parametry ještě více omezí množinu obsahující vybrané záznamy *session*. Všechny tyto nastavené parametry jsou sestaveny do formátu JSON a odeslány na server, kde jsou vyhodnoceny. Výsledkem dané funkce je číslo v procentech, které udává zastoupení sledovaných událostí ve zvoleném základu.

Kapitola 5

Serverová část

V této kapitole bude popsán zvolený způsob komunikace v systému a návrh databáze. Také budou popsány výhody tohoto řešení, návrh celé serverové aplikace a implementace jednotlivých funkcí.

5.1 Práce s daty a komunikace

Pro využití serveru pro ukládání dat a jeho výpočetního výkonu je nutné definovat rozhraní, které bude využíváno ke komunikaci klient-server. Pro způsob komunikace, kterou vždy inicializuje klient, byl určen HTTP / HTTPS požadavek na straně klienta a příslušná odpověď. Pro tento systém byla zvolena REST architektura, která je jednoduchá, implementačně nenáročná a pro tento způsob komunikace dostačující. Základní myšlenkou této architektury je manipulace s daty pomocí 4 operaci CRUD – Create, Read, Update, Delete. REST architektura je bezestavová, tzn. každý požadavek musí obsahovat všechny informace, které jsou nutné k jeho vykonání. Systém bude ve všech případech své komunikace využívat data zapsaná ve formátu JSON. Samotné rozhraní serveru bude implementováno v jazyce PHP.

Při požadavku klienta jsou data parsována z formátu JSON podle požadované operace a na jejich základě jsou provedeny odpovídající operace. Při zaslání sbíraných dat z mobilního zařízení nebo vytvoření nové verze aplikace při importu knihovny jde o jednoduchou operaci vložení. V případě ověření přihlašovacích údajů nebo získání údajů o sledovaných aplikacích je aplikován jednoduchý dotaz, který je proveden nad malým množstvím záznamů v tabulkách.

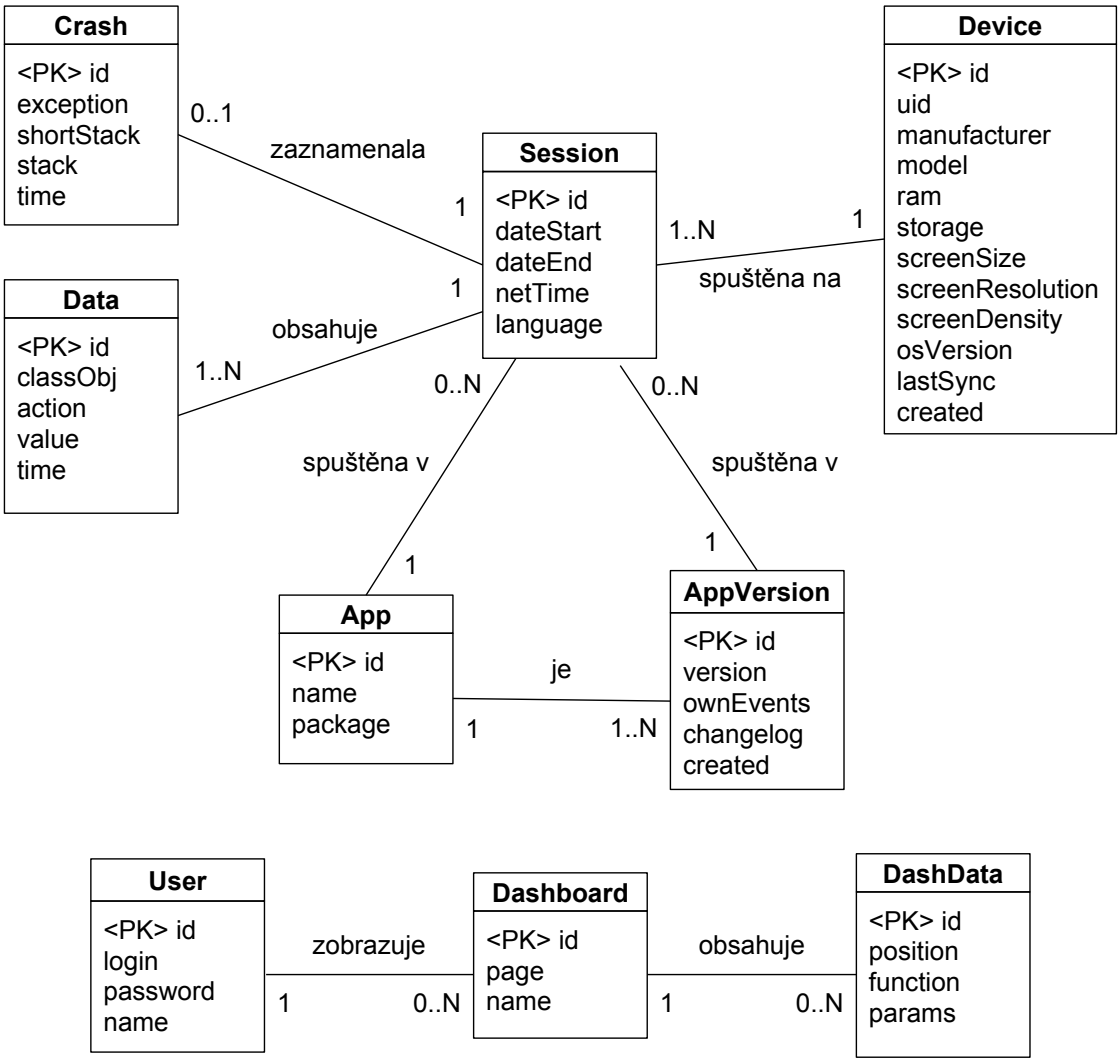
Složitějšími operacemi budou volání funkcí, které budou pracovat s obrovským množstvím dat. Tyto operace mohou být složeny z několika dotazů a data dále zpracovávány v rámci PHP skriptu. Tento proces by v některých případech mohl vézt k delší době zpracování, než by byl uživatel schopen akceptovat. Proto bude nutné při implementaci otestovat časovou náročnost jednotlivých funkcí. Kvůli náročnosti těchto operací byl zvolen způsob, kdy jsou veškerá data zpracovávána v rámci serveru. Tento způsob práce s daty má tyto výhody:

- vyhnutí se přenosu velkého objemu dat při komunikaci klient-server, server-klient
- vyhnutí se zpracování dat na straně klienta, který nemusí disponovat velkou výpočetní silou (pro klienta jsou tak mnohem menší hardwarové nároky)

- možnost využití jednoduchých agregačních funkcí nebo aritmetiky přímo v SQL, které umožní rychlejší zpracování dat

5.2 Návrh databáze

Pro ukládání dat bylo nutné navrhnout a implementovat tabulky databáze. Použitá databáze serveru je typu MySQL. Náhled v podobě ER Diagramu je vidět na obrázku 5.1.



Obrázek 5.1: ER diagram pro databázi serveru

Entita *Session* obsahuje data vztahující se ke konkrétní *session*, která byla zaznamenána v mobilním zařízení. Každá *session* je vázána k datům, která byla sbírána, aplikaci a její verzi.

Entita *Data* zaznamenává samotné události, které proběhly v mobilní aplikaci - charakterizuje o jakou událost šlo, jaká akce byla provedena a v jakém čase. Vázána je vždy k jedné konkrétní *session*.

Entita *Crash* je složena z informací, které vedly k pádu aplikace. Pro konkrétní *session* může nastat maximálně jeden pád. Při tomto ukončení aplikace je ukončena i *session*, která byla před pádem aktivní.

Entity *App* a *AppVersion* obsahují data o názvu aplikace, její verzi a sledovaných událostech v dané verzi aplikace. Uloženy je zde také přehled změn konkrétní verze aplikace oproti své předcházející.

Entita *Device* zaznamenává statické informace o zařízení, které je používáno během *session*. Jsou zde uloženy informace značce a typu zařízení, její obrazovce nebo verzi operačního systému.

Databáze byla navržena tak, aby bylo možné snadno rozšířit funkcionalitu aplikaci. Ta je závislá především na datech, která jsou sbírána. Pokud bychom tedy chtěli sbírat data jiného typu, lze toho snadno dosáhnout rozšířením entity *Session* o další sloupce nebo úpravou samotné knihovny, která může sbírat další provedené události. Doplnění knihovny by pak nemělo mít vliv na samotnou entitu *Data*.

Oddělenou část databáze tvoří entity *User*, *Dashboard* a *DashData*. Tyto entity obsahují informace o uživateli aplikace a jeho osobním nastavení jednotlivých dashboardů. Samotná entita *DashData* obsahuje informace o tom, která funkce má být zobrazena, s jakými parametry a pozici, kde v dashboardu má být umístěna.

5.3 Rozšíření databáze

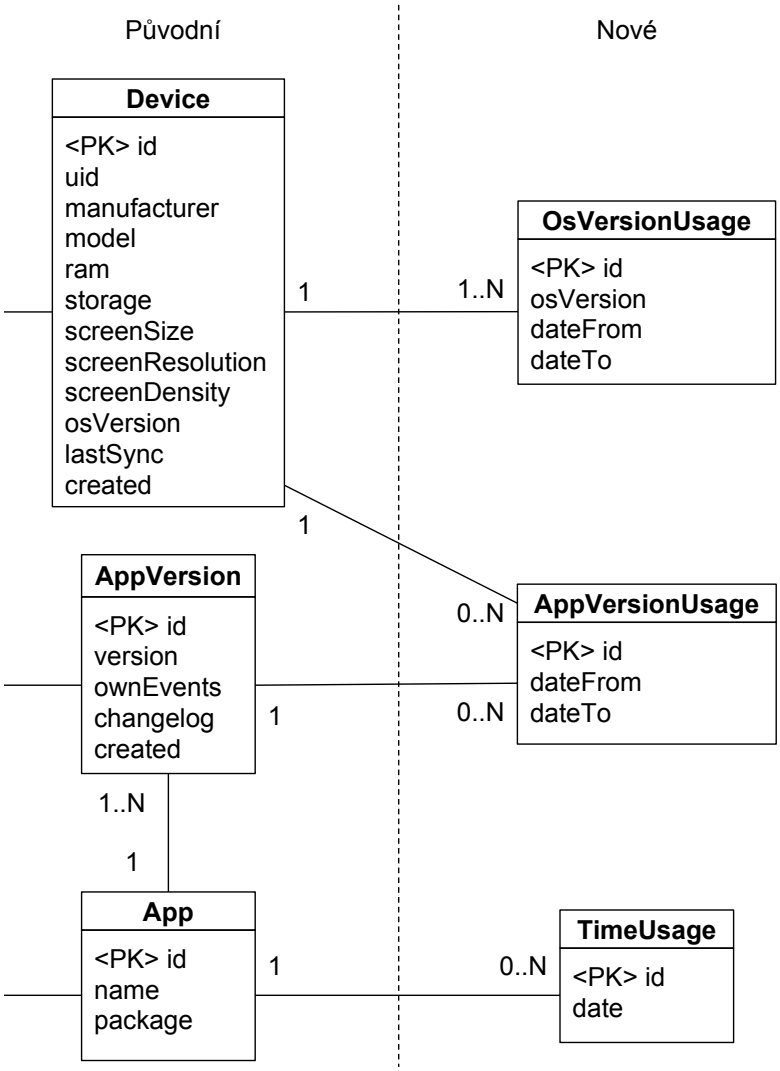
Některé informace, které jsou využívány při určitých funkcích, by ze stávající struktury entit bylo náročné získat. Například pro funkci časového využití by musely být prohledávány všechna data entity *Session* a navíc poté v cyklech zpracovávány pomocí dalších funkcí. To by z časového hlediska byly poměrně náročné operace. Proto bylo vytvořeno řešení, které využije další entitu *TimeUsage*. Do té bude při zápisu ukládán čas, ve kterém byla aplikace využívána. Tento zápis bude prováděn ihned při vkládání dat zaslaných z mobilního zařízení. Získávání výsledných dat bude tak snadnější. Schéma rozšíření ER diagramu reprezentujícího databázi serveru je zobrazeno na obrázku 5.2.

Podobně je využita entita *OsVersionUsage* pro získávání informací o používané verzi operačního systému v určitý den. Entita obsahuje informace o verzi systému a v jakém časovém horizontu byla používána. Stejným způsobem je použita entita *AppVersionUsage* pro získávání informací o používání konkrétní verze aplikace v určitý den. V této entitě je také uložen interval, v kterém byla tato verze používána. Pro obě entity je využívána hodnota `0000-00-00 00:00:00` ve sloupci *dateTo* pro dosud neukončený interval. Pokud dojde k aktualizaci systému nebo aplikace, je ukončen interval záznamu v příslušné tabulce a vytvořen záznam nový.

5.4 Implementace

V této části je popsáno, jak bylo postupováno při implementaci serverové aplikace. Jsou zde představeny jednotlivé funkce, kterými server disponuje.

Adresářová struktura byla rozdělena do tří částí, které odpovídají rozhraním pro knihovnu komunikující z mobilní aplikace (adresář *library*), modulu pro import této knihovny (adresář *import*) a samotných funkcí, které komunikují s klientskou aplikací (adresář *app*). Všechny části musí mít správně uloženou adresu pro zasílání požadavků na server. Základní webový adresář, který využívá celá desktopová aplikace, včetně modulu pro import, je za-



Obrázek 5.2: Schéma rozšíření databáze

dán na úvodní obrazovce při přihlašování. Modul knihovny má uloženou adresu serveru v řetězci své třídy, který byl vytvořen při jejím importování do mobilní aplikace.

5.4.1 Přidávání nových session

Modul knihovny využívá pro svou komunikaci se serverem adresář *library*. Knihovna využívá server k jediné funkci a tou je zapsání proběhlých *session* do databáze. Tuto funkci zajišťuje skript *add.php* v odpovídajícím adresáři. *Session* mohou být ze zařízení odesílány hromadně. Skript tedy přijímá pole objektů ve formátu JSON, které postupně projdou procesem zápisu do databáze. Proces je složen z několika kroků.

1. Ověření, zda dané zařízení je uloženo v databázi.
   Pokud je informace o zařízení součástí databáze, je zjištěno, zda je zaznamenána aktuální verze systému, případně je tento záznam upraven a zaznamenány jsou také

změny v tabulce *OsVersionUsage*. V případě, že zařízení vůbec není součástí *Device*, pak jsou vytvořeny nové záznamy o zařízení.

2. Uložení *session*.
   Do tabulky *Session* je uložen záznam o ukládané *session*, ve které je zaznamenáno použité zařízení, interval jejího běhu a další informace.

3. Uložení informace o pádu aplikace.
   Pokud během dané *session* došlo k pádu aplikace, pak jsou informace, které vedly k jejímu pádu také zaznamenány do tabulky *Crash*.

4. Uložení dat o chování uživatele.
   Všechna data, která byla posbírána s využitím automatického sběru pomocí komponent knihovny nebo pomocí statické metody `LogEvent.event` jsou zapisovány do tabulky *Data*.

5. Uložení dat o časovém využívání aplikace.
   Funkce `insertTimeUsage` analyzuje interval používání aplikace a pro každou hodinu, která je v intervalu obsažena, je proveden záznam do tabulky *TimeUsage*.

6. Uložení dat o používané verzi aplikace.
   Funkce `insertAppVersionUsage` ověří, zda je v databázi zaznamenána aktuálně používaná verze systému. Pokud není, je zaznamenáno konečné datum pro poslední verzi a vytvořen nový záznam.

5.4.2 Komunikace s modulem pro import knihovny

Komunikace serveru s modulem pro import probíhá ve dvou fázích. Pokud se jedná o aktualizaci aplikace, pak první komunikace probíhá v prvním kroku importu, kdy jsou staženy informace o předchozích verzích. Součástí dotazu klienta je identifikátor aplikace, která prošla aktualizací. Informace jsou využity při zobrazení změn, které byly klíčové pro předchozí verze v druhém kroku importu. Druhá fáze komunikace probíhá při posledním kroku, kde jsou prováděny změny. Klient zasílá informace a nové verzi aplikace, případně celé nové aplikaci a odpovídající skript `newVersion.php`, resp. `newApp.php` provádí vložení dat do tabulek *AppVersion*, resp. *App*.

5.4.3 Dashboard

Funkce dashboardu je rozdělena, v rámci komunikace se serverem, na dva typy. Data každé záložky dashboardu jsou plněny na základě informací v tabulce *DashData*. Na základě těchto dat jsou vytvářeny požadavky pro funkci, která má být zobrazena. Toto chování zajišťuje skript `getDashboard.php` a jeho funkce `doRequest`, která zajišťuje volání ostatních funkcí. Poté všechny data, které mají být zobrazeny na zvolené záložce, jsou hromadně přenesena klientovi. Druhým typem komunikace jsou operace se záložkami dashboardu a jejich grafy. U obou je umožněno jejich vytváření a mazání. Tyto operace jsou zprostředkovány skriptem `dashboardOperation.php`, který vhodně vkládá nebo maže záznamy v tabulce *DashData* a *Dashboard*.

5.4.4 Pády aplikace

Uživatel na desktopové aplikaci zvolí parametry, které chce použít při vyhledávání pádů aplikace. Ze zvolených parametrů jsou vytvářeny složené podmínky, které jsou poté využity v rámci klauzule *WHERE* v SQL dotazu databáze. Detailní pohled v desktopové aplikaci umožňuje, kromě dat o pádu aplikace, zobrazení informací o dané *session*, zařízení i aplikaci. Tudíž byl vytvořen dotaz, kde budou vhodně spojeny tabulky pomocí klauzule *JOIN*.

5.4.5 Časové využití

Funkce časového využití je zpracovávána skriptem `time_usage.php`, který má dva vstupní parametry. Prvním parametrem je počet dní, který určuje velikost voleného intervalu od současného data do minulosti. Druhým parametrem je identifikátor aplikace, u které chceme získat data o jejím využití. Výsledkem této funkce je histogram použití aplikace v určitých hodinách dne reprezentovaný polem o velikosti 24. K získání dat je využita tabulka *TimeUsage* a dotaz zobrazený na obrázku 5.3.

```
SELECT HOUR(date) as hour, COUNT(*) as value
FROM TimeUsage
WHERE appId=$appId AND (date BETWEEN '$start' AND '$end')
GROUP BY HOUR(date);
```

Obrázek 5.3: SQL dotaz pro získání časového využití aplikace

V tomto dotazu jsou použity proměnné – `$appId` je identifikátor aplikace, `$start` a `$end` jsou data intervalu, pro který jsou data zjišťovány. Důležité pro tuto funkci je ukládání dat, kdy jeden záznam představuje použití aplikace v určité hodině. Klíčové pro tuto funkci je využití funkce `HOUR`, která z data `date` získá hodnotu hodiny. Agregační funkce `COUNT` spočítá počet záznamů, které jsou seskupené právě podle hodiny. Výsledek už není nutné dále zpracovávat, pouze je vložen do pole v odpovídajícím pořadí.

5.4.6 Využití dat ze zařízení

Využití dat ze zařízení je skupina funkcí, které jsou vázány ke statickým informacím o daných zařízení v určitém časovém okamžiku. Jedná se o funkce, které zobrazují počty zařízení podle verze jejich OS, verze aplikace, jejich značky či modelu a velikosti displeje. Vstupní parametry těchto funkcí jsou vždy datum z minulosti až do současnosti a identifikátor aplikace. Výsledky jsou formátovány do seznamu dvojic, kde dvojice je složena z identifikátoru (například název značky zařízení nebo verze aplikace u odpovídajících funkcí) a čísla, které reprezentuje počet výskytů daného identifikátoru.

```
SELECT Device.manufacturer, COUNT(DISTINCT Device.id) as value
FROM Device
JOIN Session ON Session.deviceId = Device.id
WHERE Device.created<='$date' AND Session.appId=$appId
GROUP BY Device.manufacturer;
```

Obrázek 5.4: SQL dotaz pro získání přehledu značek zařízení

Příkladem je SQL dotaz zobrazený na obrázku 5.4. Tento dotaz umožňuje na základě záznamů o zařízeních v tabulce *Device* zobrazit jaké značky zařízení jsou pro danou aplikaci používány. Klíčové pro tento dotaz je spojení funkce `COUNT` s klauzulí `DISTINCT`, které umožní v rámci SQL dotazu spočítat výskyt konkrétních zařízení.

5.4.7 Prohlížení session

Funkce prohlížení *session* je rozdělena na dvě fáze. V první fázi jsou přijata data od klienta. Skript `browse_session.php` zpracovává tuto funkci, která má několik vstupních parametrů. První určuje délku intervalu, v kterém se mají *session* nacházet. Pokud se jedná o zobrazení *session* podle konkrétního data, pak je skript zpracovává podle druhého parametru. Tyto parametry nejsou nikdy zadány současně. Možnost zobrazení výhradně *session*, které skončily pádem aplikace, je indikován pomocí třetího parametru. Data jsou zpracovány pomocí SQL dotazu a vrací klientovi odpověď, která obsahuje pouze data pro zobrazení tabulky. Až při požadavku o zobrazení detailního zobrazení vybrané *session* jsou zaslána její data a relevantní informace o zařízení, či pádu aplikace.

5.4.8 Sledování událostí

Funkce sledování událostí je zpracovávána skriptem `track_event.php`, který má několik vstupních parametrů. První dva parametry představují časový interval, ve kterých byly vybrané *session* zaznamenávány. Dalšími parametry jsou identifikátory aplikace a její verze. Předposledním parametrem jsou zadané údaje pro výpočet základu funkce. Všechny tyto parametry jsou postupně sestavovány do SQL dotazu, kde pomocí kritérií klauzule `WHERE` a klauzulí `COUNT` a `DISTINCT` zjistí počet *session*, jejichž data vyhovují zadaným kritériím.

Posledním parametrem je pole údajů událostí, které chceme sledovat. Zjištění počtu *session*, které vyhovují i této skupině parametrů bude provedeno stejným dotazem, avšak kritéria klauzule `WHERE` budou rozšířena o podmínky vytvořené z posledního parametru. Tímto je získán počet *session*, které obsahují sledované události. Výsledek této funkce ukazuje zastoupení sledovaných událostí vůči základu. Je tak získán podílem jejich četností.

Kapitola 6

Testování a vyhodnocení požadavků

V této kapitole bude popsána testovací mobilní aplikace, která byla vytvořena a využita během testování systému. V rámci této části budou zároveň vyhodnocena některá sbíraná data a zobrazena v grafech. Závěr kapitoly bude zaměřen na vyhodnocení požadavků na systém, které byly stanoveny v kapitole 2.2.

6.1 Testovací aplikace a testování

Tato část je zaměřena na popis mobilní aplikace, která byla vytvořena za účelem otestování správnosti implementace systému. Pro tento účel byla navržena a implementována aplikace, která je formou dotazníku. Aplikace obsahuje několik aktivit, kde jsou umístěny různé grafické komponenty, které podporuje modul knihovny. Tato aplikace prošla procesem importu modulu knihovny a její komponenty jsou tak schopny automatizovaně sbírat informace o jejich vyplňování. Data jsou evidována pomocí komponent knihovny a také pomocí vlastních událostí.

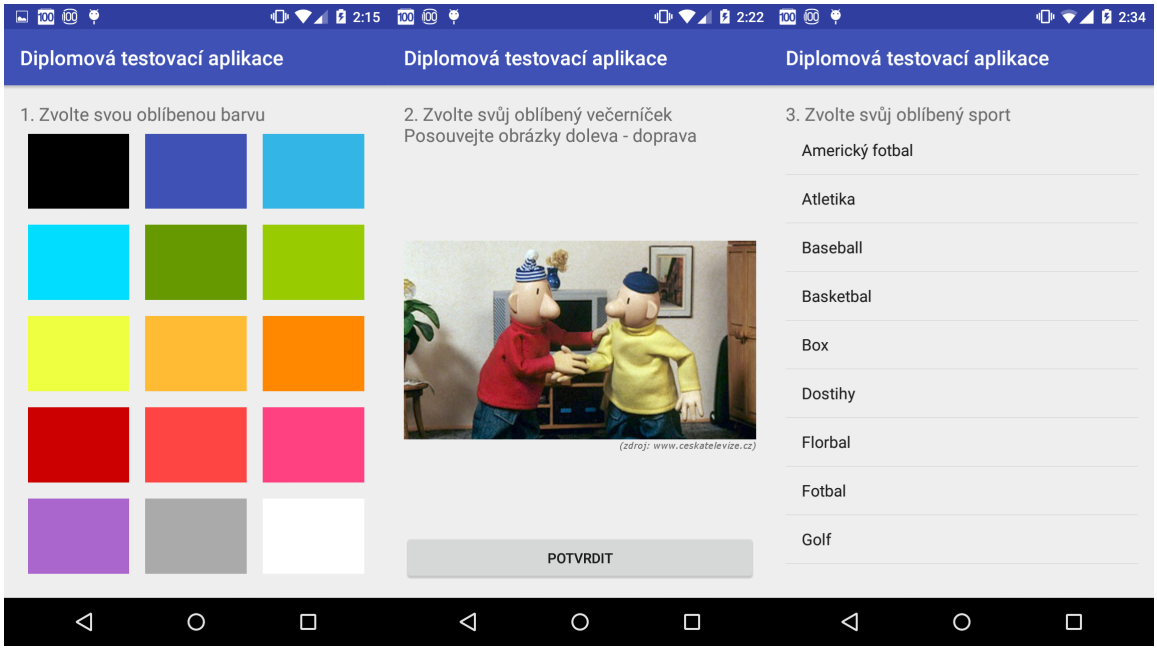
6.1.1 Testovací aplikace

Testovací aplikace je složena z několika aktivit, které zde budou postupně popsány. Každá aktivita obrazovek 1-5 obsahuje jednu otázku, na kterou uživatel odpoví a zodpovězením se přesune na otázku další. Po těchto otázkách je nabídnuta možnost vyplnění dobrovolné části dotazníku, který představuje aktivitu obrazovky 6. Náhledy obrazovek jsou na obrázcích 6.1 a 6.2.

První aktivita je spojena s výběrem oblíbené barvy uživatele. Tyto barvy jsou reprezentovány tlačítky s barevným pozadím. Bylo vybráno několik barev a vytvořeny objekty třídy `ImageButton`. Každý objekt má svůj identifikátor, podle kterého jej lze rozlišit a název vždy odpovídá konkrétní barvě, aby bylo možné zjistit, jaké barvy jsou využívány. Identifikátor je totiž jediný údaj, který je u komponenty `ImageButton` sbírán podle tabulky 3.1.

Druhá aktivita prezentuje otázku na výběr oblíbeného večerníčku. Pomocí komponenty `ViewPager` jsou zobrazeny obrázky z vybraných večerních pohádek. Po výběru stránky je potvrzena volba stisknutím tlačítka *Potvrdit*. Po potvrzení je volána metoda pro sledování vlastních událostí a uloží informaci o právě vybrané pohádce.

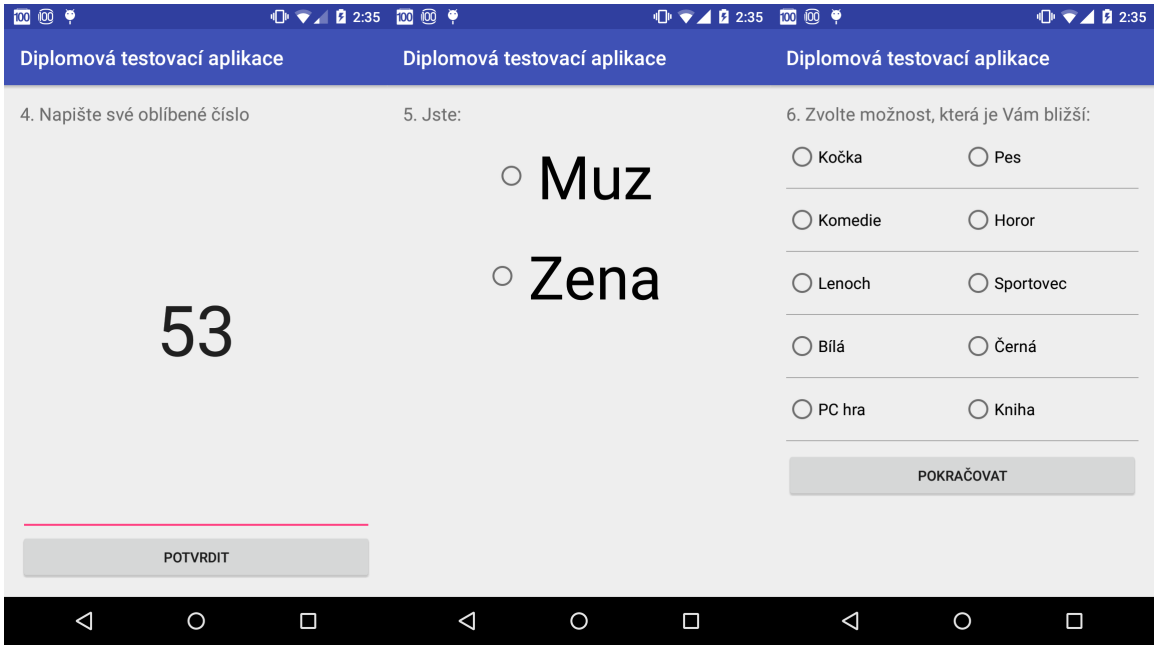
Třetí aktivita je spojena s volbou oblíbeného sportu. Pomocí komponenty `ListView` je zde zobrazen jednořádkový seznam vybraných sportů. Po kliknutí na konkrétní sport jsou



Obrázek 6.1: Náhled obrazovek 1, 2 a 3

uloženy informace o vybrané položce seznamu – její textový obsah, identifikátor a pozici v seznamu.

Čtvrtá aktivita požaduje zvolení oblíbeného čísla. Tato volba je prováděna pomocí komponenty `EditText`. Po zadání čísla a přechodu na další krok je hodnota z textového pole uložena.



Obrázek 6.2: Náhled obrazovek 4, 5 a 6

Pátá aktivita umožňuje jednoduchou volbu pomocí komponenty `RadioButton`. Uživatel zvolí, zda je muž, či žena a touto volbou se přesune na další obrazovku. Na ní je mu nabídnuta možnost ukončení dotazníku nebo pokračování. Pokud se rozhodne uživatel dotazník ukončit, přesune se na poslední aktivitu aplikace.

Šestá aktivita nemusí být nutně vyplňována v každé *session*. Slouží k demonstraci ukázky přístupu do volitelné aktivity aplikace. Obsahuje několik jednoduchých voleb mezi dvěma možnostmi, kdy uživatel vybírá tu, která je mu bližší. Tyto volby jsou prezentovány pomocí komponent `RadioButton`.

V poslední aktivitě je vložen zdrojový kód pro ukončení právě probíhající *session* a zahájení odesílání dat na server. Tato úprava byla zvolena pro okamžité odeslání dat na server, aby mohlo být provedeno vyhodnocení výsledků v nejbližší možné době.

6.1.2 Testování

Aplikace prošla procesem importu bez problémů – všechny části byly úspěšně provedeny. Aplikace byla poté bez dalších úprav vystavěna do instalačního balíčku formátu *apk* pomocí vývojového prostředí Android Studio. Tento instalační balíček byl distribuován 17 testovacím uživatelům, kteří danou aplikaci nainstalovali na mobilní zařízení a vyplnili dotazník. Pro demonstraci funkčnosti byly vytvořeny ukázkové případy, jak lze využít sbíraná data.

Oblíbený večerníček

Cílem otázky druhé aktivity bylo zjistit, který z pěti večerníčků v nabídce je nejpopulárnější. Odpovědi byly zaznamenávány pomocí metody pro ukládání vlastních událostí, která byla vyvolána při stisknutí potvrzovacího tlačítka. Vyhodnocení proběhlo s využitím funkce pro sledování událostí. Pro zjištění procentuálního zastoupení každé odpovědi bylo nutné zadat postupně všechny možnosti odpovědí. Pro vyhledání pomocí parametrů byly vyplněny pole třída, akce a parametr 1. Použité hodnoty byly klíčové slovo *ownEvent*, identifikátor vlastní události *vecernicek* a název hledaného večerníčku v odpovídajícím pořadí. Dosažené výsledky jsou zobrazeny v tabulce 6.1.

Večerníček	**Procentuální zastoupení**
Krkonošská pohádka	23,53 %
O Krtkovi	29,41 %
Bob a Bobek - králíci z klobouku	17,65 %
Pat a Mat	29,41 %
Maxipes Fík	0 %

Tabulka 6.1: Oblíbenost vybraných večerníčků

Nepovinná část dotazníku

Tato sekce je cílená na zjištění relativní četnosti přístupu do určité části aplikace. Pro ukázku využití byla zvolena šestá aktivita, která představuje nepovinnou část dotazníku. Pro zjištění četnosti přístupů do této části je nutné zadat název třídy aktivity – `Screen6` v rámci funkce pro sledování událostí. Takto je zjištěn počet přístupů (v rámci jedné *session*

je započítán maximálně jeden přístup) do dané aktivity a je porovnán vůči všem zaznamenaným *session* v databázi. Celkem bylo zaznamenáno 94,12 % přístupů do této aktivity.

Porovnání volby PC hra vs. kniha

Tato otázka byla zodpovídána v rámci nepovinné části dotazníku. V tomto případě bude opět nutné využít funkce pro sledování událostí. Při tomto porovnávání však nechceme využít relativní četnosti porovnávané vůči všem záznamům *session*, ale pouze oproti těm, ve kterých bylo vstoupeno do šesté aktivity. Proto je využito volby základu funkce, kde je zadána třída aktivity – `Screen6`. V části pro zadání parametrů vyhledávání jsou poté zadány požadované odpovědi a postupně zjištěny jejich četnosti vůči počtu vstupů do nepovinné části dotazníku. Výsledky této otázky jsou zobrazeny v tabulce 6.2.

Volba	**Procentuální zastoupení**
PC hra	43,75 %
Kniha	50 %
Nevybráno	7,25 %

Tabulka 6.2: Porovnání volby PC hra vs. kniha

Do nepovinné části dotazníku vstoupilo 94,12 % uživatelů, což odpovídá 16 uživatelům. Vůči tomuto počtu byly porovnávány volby PC hra a kniha. Součet těchto relativních četností není roven 100 %, protože zbylých 7,25 % uživatelů nevybralo žádnou z těchto možností a ukončili dotazník.

Statistika zařízení podle OS

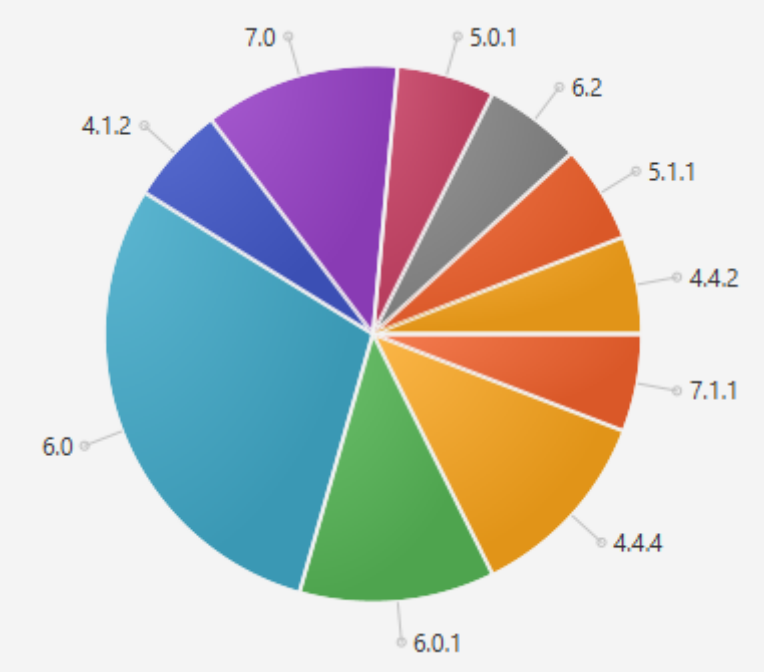
Tato statistika ukazuje, jak jsou zastoupeny různé verze systému u uživatelů testovací aplikace. Při větším vzorku zařízení můžou výsledné informace ovlivnit rozhodování pro další vývoj – může být směřován na novější verze OS a omezeno využití zpětně kompatibilních komponent. Uživatelé testovací aplikace používají asi ze 65 % operační systém Android 6.0 a novější. To může také upřednostnit vývoj funkcí spojených s novějšími verzemi OS – například využití čtečky otisku prstů. Rozložení OS je možné vidět na obrázku 6.3.

6.2 Vyhodnocení požadavků na systém

Tato část hodnotí jednotlivé požadavky specifikované firmou Nextis s.r.o. při návrhu této aplikace.

6.2.1 Univerzálnost a snadná aplikovatelnost

Tento požadavek byl vytvořen, aby bylo možné systém automaticky vložit do libovolné aplikace běžící pod OS Android. Podařilo se vytvořit systém, který pracuje se strukturou definovanou vývojovým prostředím Android Studio, což je oficiální vývojové prostředí pro vývoj aplikací pro OS Android. Systém je otestován pro struktury aplikací vytvářené tímto



Obrázek 6.3: Zastoupení jednotlivých OS na zařízeních používaných testovací aplikací

prostředím od začátku implementace až po její ukončení[1]. Pro takto vytvořené aplikace je systém funkční a plně automatický.

6.2.2 Oddělitelnost verzí jednotlivých aplikací

Jednotlivé verze jsou zaznamenávány odděleně tak, aby je bylo možno rozlišit v rámci různých funkcí – například vyhledávání pádů aplikace podle určité verze aplikace nebo přímo zobrazení využití jednotlivých verzí aplikace ke zvolenému dni. Kromě těchto funkcí jsou při importu zaznamenávány úpravy, které obsahují jednotlivé aktualizace.

6.2.3 Sledování běhu mobilní aplikace

Hlavní myšlenkou tohoto požadavku byla možnost automaticky zaznamenat významné kroky při používání mobilní aplikace. Toho bylo docíleno pomocí importu tříd, které obsahuje knihovna. Tyto třídy byly nahrazeny ve zdrojových kódech původní aplikace, ale na výslednou aplikaci nemají žádný dopad. Kromě tohoto automatického způsobu, umožňuje systém sledování vlastních událostí a to pomocí jediného volání metody.

6.2.4 Sledování závislostí na různých stavech

Systém umožňuje sledování v závislosti na denní době. Tato funkce umožňuje zobrazit, v jakou hodinu je aplikace jakou mírou používána uživateli. Sledování závislostí na dalších stavech musí být řešeno v rámci návrhu a implementace individuálně. Dále jsou sbírány data o využívání WiFi / mobilní sítě k přístupu k internetu, avšak zatím nebyly vytvořeny funkce pro jejich vyhodnocení.

[1]Vývoj probíhal od prosince 2016 do května 2017 – verze používané v tomto období byly 2.2.3, 2.3.0, 2.3.1 a 2.3.2

6.2.5 Přehledné zobrazení informací

Cílem bylo vytvořit přehledné a intuitivní zobrazení dat pomocí grafů a tabulek, aby byly vhodné podle zvolené funkce. Grafické výstupy všech funkcí byly diskutovány s odpovědnými osobami firmy Nextis s.r.o a po dohodě s nimi byly zvoleny implementované způsoby zobrazení dat jako nejvhodnější pro další práci se systémem.

Kapitola 7

Závěr

Cílem práce bylo vytvořit systém pro sledování chování uživatelů mobilních aplikací, který by umožnil lepší vývoj těchto aplikací. Hlavní přínos je zejména v možnosti lepšího rozvoje a přizpůsobení aplikací na základě analýzy chování stávajících uživatelů. Sekundární přínos je funkce pro sledování pádů aplikace, která zaznamenává vynucené ukončení aplikace systémem a chybový kód, který ho způsobil. Kromě toho nabízí systém zobrazení informací o zařízení, které využívají aplikace s importovaným sledovacím systémem, tudíž je například možné zjistit, na jakou verzi systému je nutné se soustředit při vývoj nebo pro jaký typ zařízení (smartphone, phablet, tablet).

V úvodu práce byla provedena analýza již existujících nástrojů pro sledování běhu aplikací a vyhodnoceny jejich klady a zápory. Podle požadavků firmy Nextis s.r.o. a této analýzy byly sepsány požadavky na systém. Na základě specifikace byla vytvořena struktura systému, návrh databáze a formát komunikace mezi desktopovou aplikací, modulem knihovny a serverovou částí. V hlavní části práce jsou popsány podrobně návrhy všech částí systému a postup při jejich implementaci. V závěru práce bylo provedeno a popsáno testování ukázkové mobilní aplikace a vyhodnoceno splnění požadavků na systém.

Součástí práce je i několik funkcí, které nebyly přímo součástí zadání, ale byly vytvořeny nad rámec zadání. Povedlo se vytvořit dashboard, který by měl umožnit uživateli lepší přístup k častěji používaným funkcím. Dále byly vytvořeny funkce, které využívají data o zařízeních uživatelů – například informace o operačním systému nebo používané verzi mobilní aplikace. Ty by měly sloužit k lepšímu přehledu o cílové skupině uživatelů aplikace. Všechny tyto funkce jsou součástí této práce již od návrhu systému a jsou popsány v odpovídajících částech kapitol.

Rozšíření systému by se mohlo týkat vylepšení funkce zobrazení pádů aplikace. Tato funkce by měla sloužit primárně k odladění chyb ve zdrojovém kódu. Pokud by vývojář aplikace na základě této funkce chybu opravil, mohl by ji označit za vyřešenou. Automaticky by pak mohly být nalezeny všechny chyby v databázi, které byly vyvolány stejným chybným kódem a i tyto záznamy označeny za vyřešené.

Modul knihovny umožňuje automatické sledování určitých komponent, které byly z hlediska vývoje mobilních aplikací vybrány za důležité. Zde se nabízí možnost rozšíření knihovny o další komponenty, které mohou být z knihoven Androidu nebo oficiální knihovny *Support Library* používány.

Literatura

[1] Appsee [online]. 2017 [cit. 2017-01-10].
URL https://www.appsee.com/

[2] CleverTap [online]. 2017 [cit. 2017-01-10].
URL https://clevertap.com/

[3] Firebase Analytics [online]. 2017 [cit. 2017-01-10].
URL https://firebase.google.com/features/analytics/

[4] Android Studio. 2017 [cit. 2017-05-23].
URL https://developer.android.com/studio/

[5] Chaffey, D.: Mobile Marketing Statistics 2016 [online]. 2016 [cit. 2016-12-21].
URL http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/

[6] Chang, L.: Have a smartphone? Chances are very high that it's running Android [online]. 2016 [cit. 2016-12-21].
URL http://www.digitaltrends.com/mobile/android-operating-system-domination/

[7] Developers, A.: Dashboards [online]. 2017 [cit. 2017-04-05].
URL https://developer.android.com/about/dashboards/index.html

[8] JetBrains s.r.o.: IntelliJ IDEA the Java IDE. 2017 [cit. 2017-05-23].
URL https://www.jetbrains.com/idea/

[9] Piejko, P.: 16 mobile market statistics you should know in 2016 [online]. 2016 [cit. 2016-12-21].
URL https://deviceatlas.com/blog/16-mobile-market-statistics-you-should-know-2016

[10] Taman, M.: *JavaFX Essentials*. Packt Publishing, 2015, ISBN 9781784398026.

Příloha A

Obsah CD

Přiložené CD obsahuje následující položky:

- **Systém pro sledování využití mobilních aplikací.pdf** - diplomová práce v elektronické podobě
- **Android Usage Tool/** - složka se zdrojovými kódy desktopové aplikace
- **Latex source/** - složka se zdrojovými kódy textu diplomové práce
- **Library/** - složka se zdrojovými kódy modulu knihovny
- **Server/** - složka se zdrojovými kódy serverové části
- **Testing mobile app/** - složka se zdrojovými kódy mobilní aplikace pro OS Android včetně importovaného systému pro sledování