Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Diplomová práce

Mobilní aplikace pro sběr experimentálních dat a metadat

Plzeň 2014 Jaroslav Hošek

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 15. srpna 2014

Jaroslav Hošek

Abstract

This master thesis deals with the development of mobile application for collection of experimental data and metadata. The reason for development of the application and theoretical background of mobile applications development are explained in the first part. Next the most widely used platforms for mobile devices and appropriate tools and systems for creating applications are described. The structure of layout is described by odML data format. The functionality requirements are summarized as well. Last part describes implementation of the application.

Abstrakt

Tato diplomová práce se zabývá vývojem mobilní aplikace pro sběr dat a metadat z experimentů. V první části jsou vysvětleny důvody vývoje mobilního klienta a teoretické pozadí nezbytné pro vývoj mobilních aplikací. Dále jsou zmíněny nejrozšířenější platformy pro mobilní zařízení a vhodné nástroje a systémy pro tvorbu aplikací. Struktura šablon je popsáná formátem odML. Též jsou zpracovány požadavky na funkcionalitu aplikace. Závěrečná část je věnována implementaci aplikace.

Obsah

1 Úvod

Požadavek na diplomovou práci, mobilního klienta pro elektrofyziologické databáze, vznikl na popud neurovědeckých organizací. Aktuálně existuje mnoho organizací a každá z nich má své vlastní nástroje a databáze pro manipulaci s naměřenými daty. Značné komplikace nastávají v okamžiku, kdy uživatel potřebuje přistupovat ke globálním datům, neboli získávat informace napříč jednotlivými pracovišti. Mnoho dnešních komerčních nástrojů pro záznam a sdílení dat je založeno na vysoce individuálních datových formátech a proprietárních softwarových nástrojích pro získání a analýzu dat. Přístup do jednotlivých databází je tedy velice nejednotný a každá platforma má svá specifika. Řešením, jak už jsem výše napsal, je vytvoření mobilní aplikace, která umožní propojit současné různorodé systémy, tím získávat a sdílet potřebné výsledky experimentů a ostatní materiály napříč jednotlivými pracovišti. Nebudou tak kladeny náročné požadavky na hardwarovou a softwarovou výbavu, postačí jen tzv. chytrý telefon.

Následující kapitola s názvem Vývoj mobilních aplikací popisuje rozdílnost jednotlivých operačních systémů, využívaných pro mobilní zařízení. V závěru kapitoly jsou shrnuty důvody volby operačního systému Android, pro tvorbu aplikace.

Kapitola Návrh šablony uživatelského rozhraní se zabývá technologií použitou pro ukládání šablon uživatelského rozhraní a definuje terminologii použitou u šablon.

Cílem práce je vytvořit aplikaci s funkcionalitou požadovanou třetí stranou, která je shrnuta v kapitole s názvem Analýza požadavků mobilní aplikace. Uživatel bude mít dvě možnosti, kam zaznamenávat výsledky výzkumu. Buď si vybere již předdefinovaný formulář z knihovny formulářů, nebo si může libovolně navrhnout svůj vlastní. Následně, může zvolený formulář "naplnit" daty, která získal z průběhu experimentu. Formulář pak může i nadále editovat, přidávat nová data, mazat či přidávat jednotlivá pole a měnit pozice již existujících polí. Následně data synchronizovat se serverem.

Předposlední část názvem Mobilní klient pro elektrofyziologické databáze se zaobírá popisem funkcionality vytvořeného mobilního klienta. Popisuje, jak je celá aplikace strukturovaná.

Poslední kapitola Testování aplikace představuje testovací scénáře mobilní aplikace. Testy jsou rozděleny na tři části. Prvním typem jsou jednotkové testy napsané pomocí knihovny Android JUnit. Dále je vytvořena sada automatizovaných testů uživatelského rozhraní pomocí nástroje Robotium. Nakonec bude uživatelské rozhraní testováno vybraným vzorkem uživatelů.

2 Definice problému

[1] V dnešní době jsme svědky prolínání rozdílných vědních oborů, díky čemuž dochází k uspokojování nároků, vzniklých díky rozvoji techniky. Příkladem je vznik neuroinformatiky. Jde o kooperaci dvou odvětví, neurověd a informatiky, kde implementujeme inovace z oblasti informatiky do neurovědeckého prostředí. Neuroinformatika výrazně ulehčuje práci vědcům. Vedle nepřeberného množství moderních analytických nástrojů existuje možnost sdílet data, získaná z konkrétních experimentů, a tak rychleji a komplexněji porozumět problematice. Bylo by velice přínosné mít potřebné informace kdykoli a kdekoli k dispozici (například na konferenci). Existuje několik organizací zabývajících se neurovýzkumem, každá z nich má své vlastní nástroje a databáze pro manipulaci s naměřenými daty. Každá organizace má jinak řešené databáze a přístupy k datům. Mnoho dnešních komerčních nástrojů pro záznam dat je založeno na vysoce individuálních datových formátech a closed-source softwarových nástrojích pro získání a analýzu dat. Tento faktor ovlivňuje ukládání a sdílení dat z neurovýzkumů značně negativně. V podstatě to zbytečně zpomaluje neurovýzkumy.

Západočeská univerzita je mimo jiné partnerem organizace: NIF (The Neuroscience Information Framework), jejíž zastřešující organizací je INCF (International Neuroinformatics Coordinating Facility).

INCF (International Neuroinformatics Coordinating Facility)[2] je mezinárodní neziskovou organizací v oblasti neuroinformatiky. Skládá se ze 17 členských zemí a dalších přidružených výzkumných skupin. Centrální pracoviště je ve Švédsku. INCF vytváří a udržuje databáze a výpočetní infrastrukturu v neurovědecké

[1] Www.neuinfo.org: why-neuroinformatics. INCF. *Www.neuinfo.org* [online]. 2012 [cit. 2014-06-18]. Dostupné z:http://incf.org/about/why-neuroinformatics

[2] incf: who-we-are. INCF. *Incf* [online]. 2012 [cit. 2014-06-18]. Dostupné z: http://incf.org/about/who-we-are

oblasti[3]. Dále vyvíjí nástroje a standardy pro mezinárodní neuroinformatickou komunitu.

Projekt "the INCF Dataspace"[4] byl vyvinut proto, aby zajistil sdílení dat (texty, obrázky, modely a simulace) mezi jednotlivými kolaborujícími výzkumnými zařízeními v oblasti neurovědy. INCF Dataspace je realizován pomocí iRODS[5], což je open-source software pro správu dat.

NIF (The Neuroscience Information Framework)[6] je webový inventář shromažďující výsledky výzkumů, měření a nástroje. Informace jsou dostupné z jakéhokoli počítače, připojeného k internetu. Vznikl v roce 2005 v USA. Mezi služby, které NIF nabízí, patří: zdroj informací a materiálů pro výzkumníky a studenty, nástroje pro výměnu dat, atd.

G-Node (German Neuroinformatics Node)[7] působí v rámci INCF a sídlí v Německu. Zaměřuje se na rozvoj technologie, infrastruktury, získávání a analýzu experimentálních dat a na vývoj open-source nástrojů pro manipulaci a analýzu neurofyziologických dat. Používá odML formát, který bude vysvětlen níže v kapitole 4.1.

CARMEN[8] je britský projekt, jehož cílem je tvorba virtuální neurofyziologické laboratoře, která umožní sdílet data, analyzovat kód a expertízy.

---

[3] What we do: http://incf.org/about/what-we-do. INCIF. *INCIF* [online]. [cit. 2014-08-14]. Dostupné z: http://incf.org/

[4] Resources: data-space. INCF. *Incf* [online]. 2012 [cit. 2014-06-18]. Dostupné z: http://incf.org/resources/data-space/access-software-and-interfaces

[5] Irods: about. RODS. *Irods* [online]. 2012 [cit. 2014-06-18]. Dostupné z: https://irods.org/about/

[6] Neuinfo: about. NEUINFO. [online]. 2010 [cit. 2014-06-18]. Dostupné z: https://www.neuinfo.org/about/index.shtm

[7] G-node: about. G-NODE. *G-node* [online]. 2014 [cit. 2014-06-18]. Dostupné z: http://www.gnode.org/about

[8] Carmen: about. CARMEN. *Carmen* [online]. 2000 [cit. 2014-06-18]. Dostupné z:http://www.carmen.org.uk/about

EEG base[9] **-** Na Katedře informatiky a výpočetní techniky[10] je vyvíjen EEG/ERP portál EEGbase. Jedná se o webovou aplikaci, která slouží komunitě výzkumníků k ukládání a sdílení dat elektroencefalografického (EEG) výzkumu. Významná část výzkumu je zaměřena na kognitivní evokované potenciály (ERP). Portál byl vyvinut jako samostatný produkt, kde přístup k databázi je zajištěn přes webové rozhraní. Systém sjednocuje rozdílné datové formáty a metadata pro ukládání EEG/ERP experimentů, které je možné spravovat, vyhledávat a sdílet.

Petr Miko[11] vytvořil v rámci své diplomové práce mobilního klienta pro portál. Nevýhodou klienta je, že mobilní telefon musí být při práci výzkumníka neustále připojen k internetu a obsahuje sadu již hotových formulářů. Základní formuláře jsou scénáře a experimenty. Scénáře obsahují popisy konkrétních průběhů prováděného experimentu a skupinu osob, podílející se na daném výzkumu. Druhý typ formuláře - Experimenty zahrnují metadata prováděného experimentu (informace o měřeném subjektu, lokalizace elektrod, software používaný při daném experimentu, scénář, atd).

Jakub Krauz vytvořil ve své diplomové práci[12] knihovnu pro vytváření šablon uživatelského rozhraní (dále jen šablona), která je nezávislá na databázové struktuře systému. Je zde podrobně popsán způsob generování šablon na serveru a interface pro jejich zpřístupnění. Spolupracovali jsme na návrhu struktury pro ukládání šablon, poněvadž mnou vytvořený mobilní klient umožňuje stáhnout ze serveru šablonu, ze které vygeneruje uživatelské rozhraní. Potřebu vytvoření knihovny odstartovala především současná situace, existence mnoha organizací (popsány výše), které mají

---

[9] Eegdatabase. ZCU. *Eegdatabase* [online]. 2008 [cit. 2014-06-18]. Dostupné z: http://eegdatabase.kiv.zcu.cz/

[10] www.kiv.zcu.cz

[11] MIKO, Petr. *Mobile system for management of EEG/ERP experiments*. Plzeň, 2013. Master Thesis. University of West Bohemia.

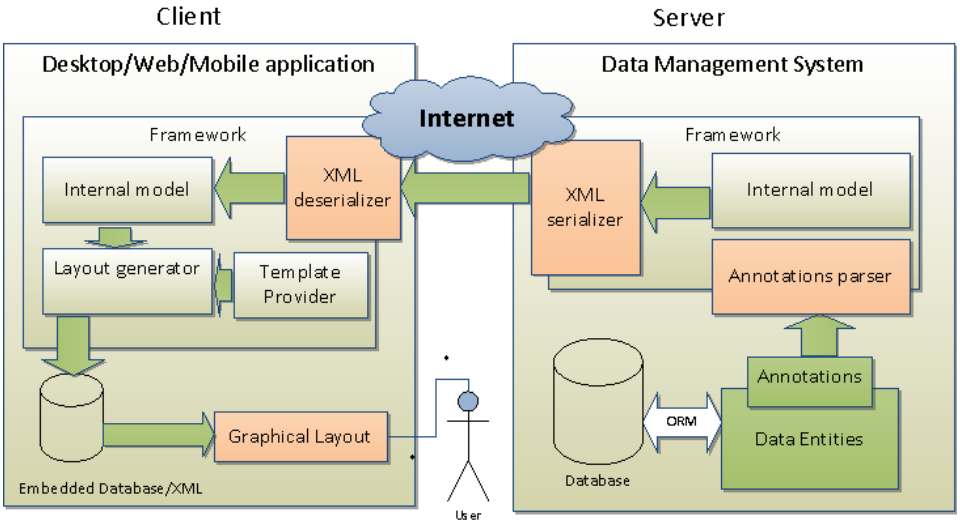
[12] KRAUZ, Jakub. *Tool for automatic generation of graphical templates of mobile devices*. Plzeň, 2014. Diplomová práce. Západočeská univerzita v Plzni.

svůj vlastní portál pro ukládání a sdílení dat z neurovědeckého výzkumu. Díky této skutečnosti zde vyvstává hned několik komplikací:

1. Každá organizace má rozdílnou strukturu dat.
2. Většinou se používají relační databáze, kde je změna struktury databáze pracná.
3. Nelze vytvořit univerzální mobilní aplikaci pro přístup k datům, aniž by nemusela být upravena.

Na obrázku 1 je schéma nástroje.

Obrázek 1 – Schéma Klient - Server[13]



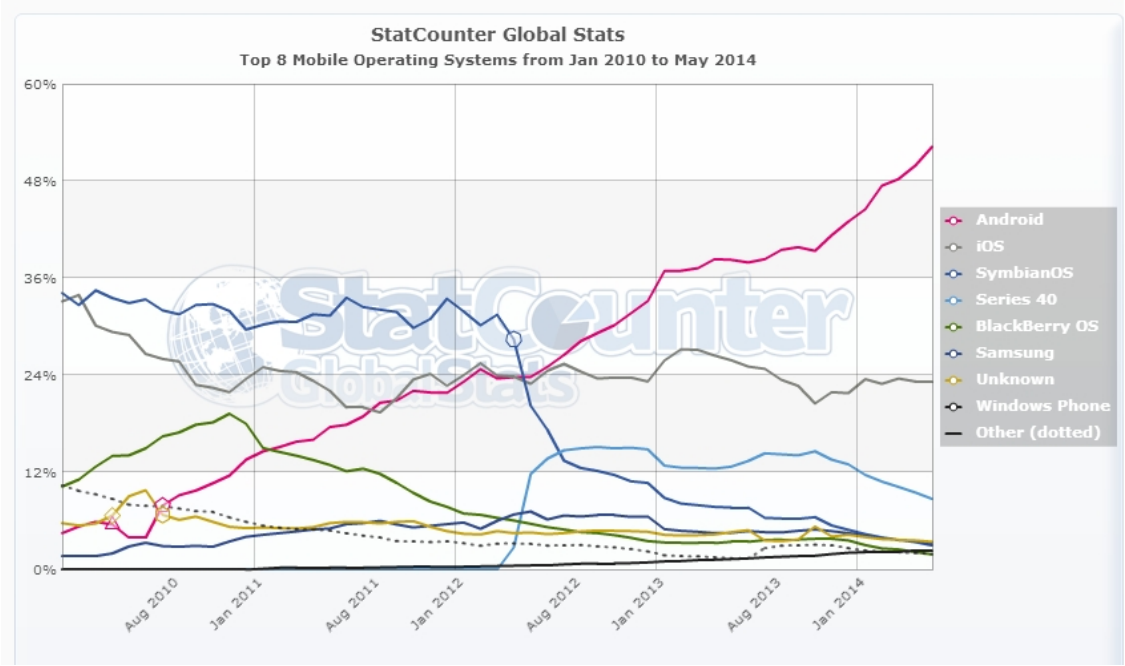
[13] PETR, Ježek a Mouček ROMAN. *Framework for automatic generation of graphical layout compatible with multiple platforms* [online]. 15-19 Sept. 2013 [cit. 2014-06-20].

3 Vývoj mobilních aplikací

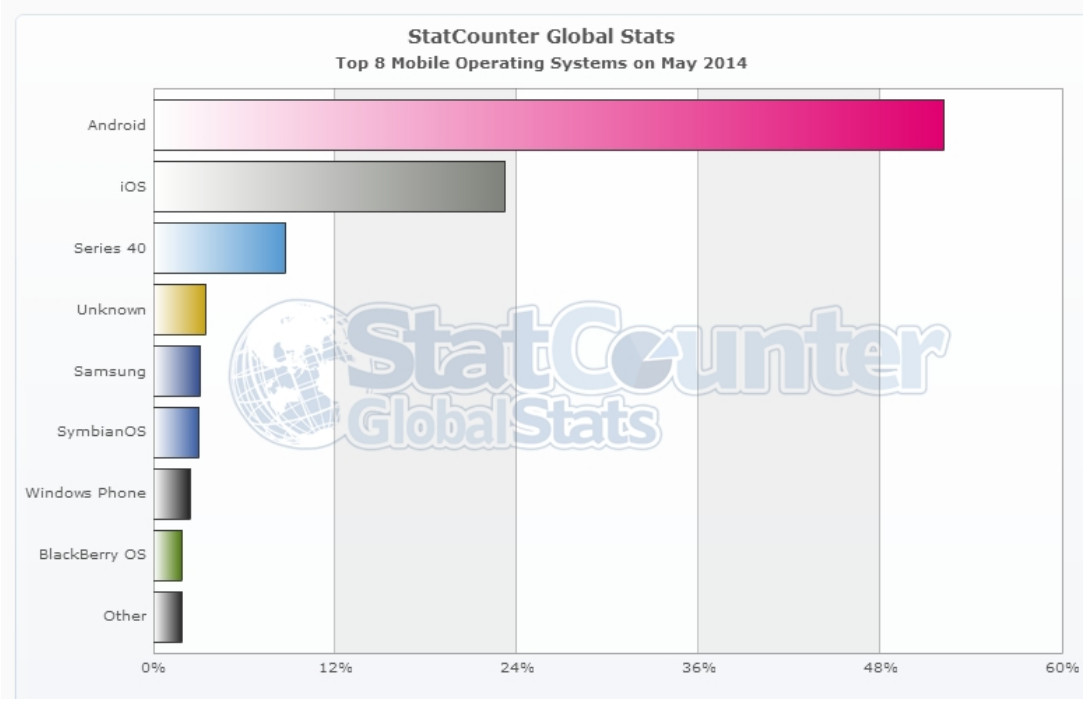
Pro srovnání platforem dle jejich rozšířenosti jsem si vybral data z nezávislého analyzačního nástroje StatCounter, který počítá zobrazené webové stránky (Page Views). StatCounter své metody aplikuje na více než tři milióny webů po celém světě.

Obrázek 2 a obrázek 3 popisují rozvoj a rozšířenost mobilních operačních systémů v čase (od začátku roku 2010 až do současnosti). Vidíme, že situace počátkem roku 2010 byla naprosto odlišná od dnešní doby. Lze říci, že stálou pozici že všech mobilních operačních systému má iOS. Sice je z časové řady patrný mírný klesající trend, nicméně křivka je ze všech ostatních nejvíce horizontální. Příčinu vidím v tom, že mobilní zařízení od společnosti Apple si drží svůj standard a kvalitu a právě díky této jedinečnosti mají stále svou věrnou klientskou základnu, která je ochotná za kvalitu zaplatit. Značně rozdílný vývoj čteme z červené, podstatně strmější křivky, která nepatří nikomu jinému, než operačnímu systému Android. Ve sledovaném období význam mobilního operačního systému Android značně nabýval na síle, hlavně díky několika faktorům. Cenová dostupnost a pestrá škála tzv. chytrých telefonů, na kterých se můžeme setkat s Androidem je velice příznivá pro širší uživatelskou základnu. Další výhoda je bezesporu velká komunita vývojářů na této platformě a snadnost publikace vytvořených aplikací. V polovině roku 2012 došlo k situaci, kdy systém Android předstihl iOS. Další, dříve nejrozšířenější mobilní operační systém SymbianOS se z trhu v podstatě vytratil. Ve své době byl nejrozšířenější právě proto, že na trhu nebyla téměř žádná konkurence, tudíž uživatelé tolerovali značné nedostatky systému.

Obrázek 2 - rozvoj a rozšířenost mobilních operačních systémů[14]



Obrázek 3 - rozvoj a rozšířenost mobilních operačních systémů[15]



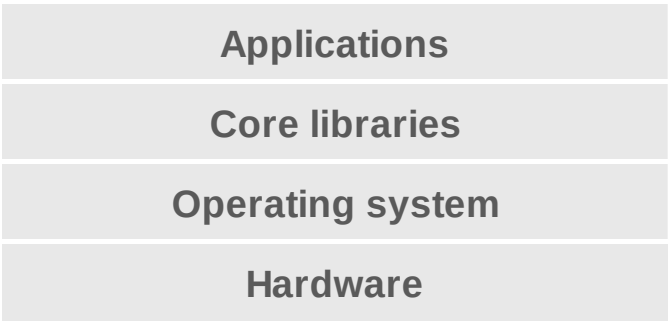
[14] *Statcounter* [online]. 1999 [cit. 2014-06-20]. Dostupné z: http://gs.statcounter.com/

[15] *Statcounter* [online]. 1999 [cit. 2014-06-20]. Dostupné z: http://gs.statcounter.com/

3.1 Základní architektura

Mobilní platformy se skládají z vrstev, které jsou uvedeny na obrázku 4. Vrstva, která se nachází na vrcholu zásobníku, se nazývá Applications a slouží ke spouštění aplikací. Pod touto vrstvou je vrstva Core libraries, která poskytuje vývojářům vytvářející mobilní aplikace sadu již hotových funkcí. Například knihovnu pro přehrávání video a audio formátů, SQLite (odlehčená relační databázová knihovna), OpenSSL, OpenGL ES (knihovna na vykreslování 3D grafiky). Operating system tvoří abstraktní vrstvu mezi používaným hardwarem a zbytkem softwaru ve vyšších vrstvách. Má na starosti například správu paměti, správu sítí, správu procesů. Poslední vrstvou je Hardware. Ta zahrnuje fyzické zařízení, neboli soubor zdrojů, ke kterým může aplikace přistupovat například obrazovka, tlačítka, gps atd.

Obrázek 4 - Obecná architektura mobilních platforem[16]



3.2 Mobilní operační systémy

Tato kapitola je zaměřena na popis operačních systémů Android (Google), iOS (Apple) a Windows Phone (Microsoft).

[16] DUFFY, Thomas J. *Programming with Mobile Applications: Android(TM), iOS, and Windows Phone 7*. UK: Cengage Learning, 2013. ISBN 978-1133628132.

3.2.1 **Android**[17]

Android je operační systém založený na upraveném linuxovém jádře, který zajišťuje zabezpečení systému, správu paměti, správu procesů, přístup k síti a ovladačům senzorů. Hardwarová stránka je podporována Open Handset Aliancí (OHA), která je konglomerátem mnoha výrobců mobilních telefonů např. Samsung, HTC, LG. Cílem OHA je vytvořit otevřené standardy pro mobilní zařízení. Software je vyvíjen Android Open Source projektem, který vede společnost Google. Zdrojové kódy Androidu jsou zveřejněny jako open-source pod ASL licencí (Apache Software License). Jádro systému je licencováno pod GPL (General Public Licence).

Android aplikace jsou psány v programovacím jazyce Java. Pro potřeby vývoje je k dispozici Android SDK (Software Developer Kit), který poskytuje nástroje a API potřebné k vývoji aplikací. Součástí SDK je i emulátor, který umožňuje testovat aplikace bez fyzického zařízení.

Veškeré aplikace jsou překládány do Java byte kódu[18], který se dále přeloží pomocí Dalvik kompilátoru do Dalvik byte kódu, jenž se nechá spustit v DVM (Dalvik Virtual Machine). DVM je vyvíjen společností Google od roku 2005 speciálně pro Android. DVM má registrově orientovanou architekturu a využívá základních vlastností linuxového jádra (správa paměti, práce s vlákny). Důvodem pro vznik DVM byla potřeba optimalizovat virtuální stroj pro potřeby mobilních zařízení, kde hlavní roli hraje výkon a úspora energie.

Vytvořené aplikace jsou spouštěné na stejné úrovni jako systémové aplikace daného zařízení. Architektura Androidu staví na předpokladu, že veškerá funkcionalita aplikační vrstvy je stejná a neznevýhodňuje aplikace třetích stran.

Mezi výhody Androidu bezesporu je existence velké komunity vývojářů. Publikace vytvořených aplikací je poměrně snadná. Vývojář zaplatí pouze jednorázový

---

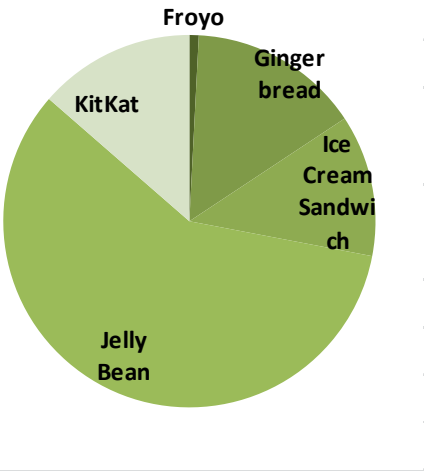
[17]Developer.android. *Developer.android* [online]. 2010 [cit. 2014-06-20]. Dostupné z:http://developer.android.com/about/dashboards/index.html

[18] VÁVRŮ, Jiří a Miroslav UJBÁNYAI. *Programujeme pro Android*. Česká Republika: Grada, 2013. ISBN 978-80-247-4863-4.

poplatek a aplikace může publikovat ihned, bez jakéhokoli čekání na schválení dané aplikace. Za nevýhody lze považovat existenci heterogenního spektra hardwarového zázemí a fragmentaci platformy (obrázek 5).

Obrázek 5 - Fragmentace Androidu[19]

Version	Codename	API	Distribution
2.2	Froyo	8	0,80%
2.3.3 - 2.3.7	Gingerbread	10	14,90%
4.0.3. - 4.0.4	Ice Cream Sandwich	15	12,30%
4.1.x	Jelly Bean	16	29%
4.2.x		17	19,10%
4.3		18	10,30%
4.4	KitKat	19	13,60%



3.2.2 Apple iOS[20]

Operační systém iOS od firmy Apple se používá v telefonech iPhone a dalších zařízeních jako je tablet iPad nebo iPod Touch. Stejně jako Windows Phone není iOS open-source.

Architektura iOS (obrázek 6) je o mnoho jednodušší než Android aplikace. Vrstva pro tvorbu aplikací se nazývá Cocoa Touch, API obsahuje komponenty k tvorbě uživatelského rozhraní, tvorbu map, kontrolery pro práci s událostmi, umožňuje odesílat zprávy atd. Vrstva Media obsahuje nástroje pro tvorbu grafiky, audia a videa. Vrstva Core services obsahuje nízko úrovňový přístup například k lokalizačním nebo síťovým službám, které využívají vyšší vrstvy, dále odlehčenou databázi SQLite a podporu pro XML.

---

[19]Developer.android. *Developer.android* [online]. 2010 [cit. 2014-06-20]. Dostupné z:http://developer.android.com/about/dashboards/index.html

[20]MCWHERTER, Jeff a Scott GOWELL. *Professional Mobile Application Development*. England: Wrox, 2013. ISBN 978-1118203903.

Důvod proč je iOS platforma jednodušší než konkurenční platformy spočívá v tom, že společnost Apple zároveň vyrábí zařízení, která iOS používají a má tedy nad vším kontrolu. Android a Windows Phone 7 musí obsahovat vrstvu s virtuálním strojem, aby mohli obsluhovat rozdílný hardware. Virtuální stroj je zodpovědný za překlad generického kódu do strojových instrukcí daného zařízení. Tato abstrakce není pro iOS potřeba poněvadž hardware je stejný na všech zařízení s iOS.

Obrázek 6 - Architektura iOS[21]

Aplikace je možné psát v jazyku C nebo pokročilejším Objective-C. Vývoj aplikací umožňuje aplikace XCode, která je dostupná pouze na operačních systémech Mac OS X.

Publikace aplikací je složitější, než u Androidu. Neplatí se jednorázový poplatek ale roční. Aby mohla být aplikace publikována, musí projít schválením, z čehož plyne, že je zde jistá časová prodleva.

3.2.3 Windows Phone

Windows Phone je mobilní operační systém firmy Microsoft. Uveden byl roku 2010. Jedná se o nástupce systému Windows Mobile, se kterým není zpětně kompatibilní.

Architektura platformy Windows Phone 7 je velice podobná architektuře Android. K vývoji lze použít z aplikační vrstvy Silverlight Framework, který umožňuje vytvářet aplikace na základě Windows Forms nebo je možné použít XNA framework

[21] DUFFY, Thomas J. *Programming with Mobile Applications: Android(TM), iOS, and Windows Phone 7*. UK: Cengage Learning, 2013. ISBN 978-1133628132.

k vývoji 2D a 3D aplikací. Frameworky je možné kombinovat dle potřeb dané aplikace. Obě technologie jsou postavené nad .NET frameworkem. Vytvářet aplikace je možné v jazyce C#, F# nebo Visual Basic .NET. Programovacím nástrojem je Visual Studio, které je pro vývoj zdarma.

Za publikaci aplikací se platí roční poplatek. Tento poplatek neplatí pro studenty, ti mohou aplikace publikovat zdarma (program DreamSpark). Vytvořená aplikace podléhá certifikačnímu procesu, který zajistí, aby splňovala podmínky dané Microsoftem.

3.3 Přenositelnost aplikací

Z předchozích kapitol je patrné, že pro jednotlivé platformy jsou využívány jiné technologie, standardy a restrikce ze strany vydavatelů mobilních platforem. Nevýhodou tedy je, že pokud chce programátor napsat aplikaci napřič platformami musí se naučit mnoho programovacích jazyků, což je značná komplikace.

Následující technologie umožní vytvořit aplikaci, tak aby byla přenositelná napříč zmíněnými platformami. Odpadá tedy problém, kdy se vývojář musí napřed naučit programovat pro každou platformu zvlášť (Java, C# a Objective-C).

3.3.1 Webové aplikace [22]

Webové aplikace jsou extrémně populárním řešením, protože je snadné je vytvořit a dále rozšiřovat. Zobrazovány jsou v mobilních webových prohlížečích, které podporují technologie pro tvorbu dynamických aplikací pomocí HTML, CSS a JavaScriptu. S příchodem HTML5 bylo přidáno mnoho funkcionality do mobilních webových prohlížečů. Například lze ukládat data na klientské straně, pokud není připojení k Internetu a jakmile dojde k obnovení spojení, tak jsou data synchronizována se serverem. V tabulce 1 vidíte seznam rozdílů standardu HTTP5 a nativních aplikací.

[22] MCWHERTER, Jeff a Scott GOWELL. *Professional Mobile Application Development*. England: Wrox, 2013. ISBN 978-1118203903.

Tabulka 1 - seznam rozdílů standardu HTTP5 a nativních aplikací[23]

	HTML5	ANDROID	IOS	BLACKBERRY	WP7
Location/GPS	Yes	Yes	Yes	Yes	Yes
Camera	No	Yes	Yes	Yes	Yes
Accelerometer	Limited	Yes	Yes	Yes	Yes
Video	Yes	Yes	Yes	Yes	Yes
Audio	Yes	Yes	Yes	Yes	Yes
Local Storage	Limited	Yes	Yes	Yes	Yes
Push Notifications	Yes	Yes	Yes	Yes	Yes
In-App Purchase	No	Yes	Yes	Yes	No
App Market	No	Yes	Yes	Yes	Yes

První výhodou mobilních aplikací je rychlost vývoje, jelikož technologii HTML ovládá mnoho vývojářů. Není třeba se učit něco nového. Dalším plusem je jednoduchost vývoje napříč platformami. Vytvoření mobilní aplikace je jednoduší, poněvadž lze sdílet jednotlivé časti kódu a v závislosti na typu aplikace pouze stačí vytvořit nové uživatelské rozhraní zapadající do zvolené platformy. Nelze opomenout ani snadnou aktualizaci aplikace, kdy vývojář může kdykoliv provést změnu, která se okamžitě projeví v aplikaci. Toto je jedna z oblíbených funkcionalit mobilních aplikací poněvadž není třeba procházet zdlouhavým publikačním procesem jako je tomu u společnosti Apple (podrobněji popsáno v kapitole 3.2.2.). Mobilní webová aplikace není svázána s pravidly obchodů jednotlivých platforem, tudíž není potřeba podstupovat proces schvalování aplikace.

Mezi nevýhody bezesporu patří fakt, že aplikaci interpretuje webový prohlížeč, což je další mezivrstva, která může mít vliv na systémové prostředky.

[23] MCWHERTER, Jeff a Scott GOWELL. *Professional Mobile Application Development*. England: Wrox, 2013. ISBN 978-1118203903.

3.3.2 Apache Cordova (PhoneGap)[24]

Apache Cordova je open source sada nástrojů vytvořených Nitobi Solutions, nyní součást společnosti Adobe, které umožní vytvářet multiplatformní mobilní aplikace založené na standardech webu (HTML, CSS, Javasccript) rozšířené o nativní funkce daného zařízení s využitím stejného zdrojového kódu mezi jednotlivými mobilními platformami. Společnost Adobe darovala PhoneGap komunitě pod Apache Foundation, která projekt přejmenovala na Apache Cordova.

Seznam dostupných API na jednotlivých mobilních platformách je znázorněn v příloze 1. Pokud některá nativní funkce chybí je možné jí vytvořit použitím nativního kódu a zavolat pomocí JavaScriptu. Seznam již vytvořených funkcí lze nalézt v Cordova Plugin repozitáři[25] .

Nyní se ještě v krátkosti zaměřím na porovnání HTML5 a PhoneGap. PhoneGap používá HTML5, tudíž přistupuje k funkcionalitě HTML5. Výhodou Apache Cordova je, že může přistupovat k hardwaru, neboli umí komunikovat se zařízením. Dále je multiplatformní, užíváme stejný javascriptový kód na všechny platformy. Aplikace je kompilovaná do nativní podoby daného zařízení a může být použita offline. V některých případech má Apache Cordova pomalou odezvu uživatelského rozhraní[26].

3.4 Shrnutí

Práci s mobilním systémem Android jsem zvolil hned z několika důvodů. Jedním z nich je, že většina studentů a akademiků používá telefony právě s operačním systémem Android, tudíž je zde široká uživatelská základna. Dalším plusem je velice

---

[24] APACHE CORDOVA. *Apache Cordova* [online]. 2012 [cit. 2014-08-15]. Dostupné z: http://cordova.apache.org/

[25] PLUGINS REGISTRY. APACHE CORDOVA. *Apache Cordova* [online]. 2012 [cit. 2014-08-15]. Dostupné z:http://plugins.cordova.io/#/

[26] MCWHERTER, Jeff a Scott GOWELL. *PROFESSIONAL MOBILE APPLICATION DEVELOPMENT*. Indianapolis, Indiana: John Wiley & Sons, Inc., 2012. ISBN 978-1-118-20390-3.

snadná publikovatelnost vytvořené aplikace. Aplikaci je možné publikovat ihned, aniž by musela projít schvalovacím procesem. Další výhodou je, že je Android open-source projektem.

4 Návrh šablony uživatelského rozhraní

Dle požadavků na strukturu šablon je třeba navrhnout univerzální strukturu sloužící k ukládání šablon uživatelského rozhraní pro sběr elektrofyziologických experimentů. Tato šablona je primárně určena pro synchronizaci mobilní aplikace se serverem. Obecné požadavky na strukturu šablon jsou:

1. Nezávislost na platformě
2. Umožnit k definici uživatelského rozhraní připojit vyplněná data
3. Snadné vytvoření šablony, tak aby ji bez problémů šlo vytvořit i na mobilním zařízení, kde jsou omezené prostředky
4. Snadná čitelnost člověkem
5. Otevřenost a dostupnost

Většina dnešních mobilních platforem má možnost ukládat uživatelské rozhraní (GUI) do XML souborů (XML - Android, XAML – Windows Phone). Bohužel jejich schéma je vytvořeno pouze pro danou platformu, čímž je znemožněna jejich přenositelnost. Jednou z možností je vytvořit si vlastní XML schéma nebo zvolit nějaký jiný formát například JSON, YAML. My jsme se rozhodli pro zvolení odML formátu. Důvody pro jeho zvolení shrnu v následující kapitole.

4.1 odML - Open Metadata Markup Language[27]

Projekt vyvíjí German Neuroinformatics Node (G-Node), podrobně popsán v kapitole 2. Důvodem vytvoření odML formátu bylo zajistit flexibilní a snadno použitelný formát pro přenos metadat.

4.1.1 Model dat

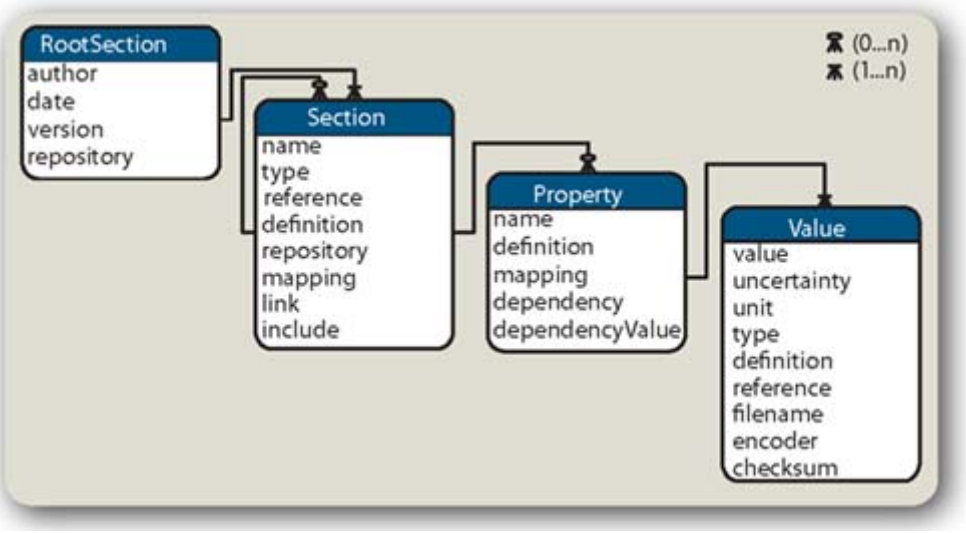
Základem formátu odML je stromová struktura (Obrázek 7 znázorňuje stromovou strukturu jako ER diagram). Model definuje čtyři entity: *RootSection (Hlavní Sekce), Section (Sekce), Property (Vlastnost) a Value (Hodnota).*

[27] Open metadata Markup Language. G-NODE. *G-node* [online]. 2014 [cit. 2014-08-15]. Dostupné z:http://www.g-node.org/projects/odml/

[28]Základní myšlenka odML modelu je založena na dvojici klíč-hodnota například „jméno=Jaroslav“. *Vlastnost* tedy může obsahovat jednu nebo více *Hodnot. Vlastnosti* jsou logicky seskupené do *Sekcí.* Rekurzivní spojení mezi *Sekcemi* umožňuje vytvořit v *Sekci* libovolný počet podsekcí. *Hlavní Sekce* představuje kořen stromu, ve kterém jsou uloženy informace o dokumentu (Autor, Datum, atd.). Uvedené entity obsahují velké množství specifických atributů, například reference, pomocí které můžeme vytvářet odkazy mimo aktuální strukturu dat. Většina z nich je pouze volitelná.

Model dat je definován obecně, nezávisle na implementaci. V současné době je podporován formát XML, kde k implementaci modelu je použito XML schéma.

Obrázek 7 - ER diagram[29]



4.1.2 Terminologie

OdML model je natolik obecný, že můžeme ukládat nebo sdílet libovolný obsah. Pokud potřebujeme data sdílet, je nutné definovat jména *Sekcí, Vlastností.* Stejně tak musíme uvést, jakého datového typu jsou *Hodnoty* a mnoho dalších atributů. Definice terminologie nám umožní standardizovat jednotlivé domény, což nám umožní efektivně

[28] Grewe J, Wachtler T and Benda J (2011) A bottom-up approach to data annotation in neurophysiology. *Front. Neuroinform.* **5**:16. doi: 10.3389/fninf.2011.00016

[29] Description of the odML data model. G-NODE. *G-node* [online]. 2014 [cit. 2014-08-15]. Dostupné z:http://www.g-node.org/projects/odml/structure

sdílet údaje se spolupracujícími systémy. OdML projekt poskytuje terminologii k neurovědecké problematice[30]. V této práci navrhneme obecnou terminologii pro ukládání uživatelského rozhraní.

Grafické komponenty jsou definovány pomocí entity *Sekce,* typ komponenty se určuje atributem *Type* a je následujícího typu: Form, Textbox, Combobox, Checkbox, Choice.

Form

Šablona musí obsahovat alespoň jednu *Sekci* typu Form, což je formulář, který obsahuje jednotlivá pole formuláře (podsekce). Jméno formuláře se zadává do atributu *Name.* Sekce s formulářem může obsahovat *Vlastnost* LayoutName, kde hodnotou je jméno šablony uživatelského rozhraní.

Formulář může obsahovat jiné podformuláře a tímto způsobem lze vytvořit strom jednotlivých formulářů. Tuto funkcionalitu je možné zobrazit jako seznam připojených záznamů z jiného formuláře. Podformulář může obsahovat *Vlastnost* (Cardinality). Při vyplňování Cardinality se zadává celočíselná hodnota, vyjadřující, kolik můžeme připojit záznamů daného podformuláře. Hodnota "-1" znamená, že můžeme připojit neomezený počet záznamů.

Názorná ukázka vytvořeného formuláře Osoba, který obsahuje podformulář:

```
<section>
    <type>form</type>
    <name>osoba</name>
    <property>
        <name>layoutName</name>
        <value>Libovolné jméno</value>
    </property>
    <section>
        <type>textbox</type>
        <name>Jméno</name>
    </section>
    <section>
        <type>textbox</type>
        <name>Datum narození</name>
    </section>
    <section>
        <type>combobox</type>
```

[30] Odml - terminologies. G-NODE. *G-node* [online]. 2014 [cit. 2014-08-15]. Dostupné z: http://www.g-node.org/projects/odml/terminologies

```
        <name>pohlaví</name>
        <property>
            <name>datatype</name>
            <value>string</value>
        </property>
        <property>
            <name>values</name>
            <value>Muž</value>
            <value>Žena</value>
        </property>
    </section>
    <section>
       <type>form</type>
       <name>Adresa</name>
      <property>
         <name>cardinality</name>
         <value>2</value>
      </property>
    <section>
       <type>textbox</type>
       <name>Město</name>
    </section>
    <section>
       <type>textbox</type>
       <name>Ulice</name>
    </section>
</section>
```

Textbox

Sekce typu Textbox, představuje komponentu umožňující zadat uživateli text do položky formuláře. Textbox může být rozšířen o vlastnosti uvedené v tabulce 2.

Tabulka 2 - Vlastnosti Textbox[31]

VLASNOST	POPIS
datatype	Určuje datový typ vstupního pole (String, Integer, Date, Email)
minLenght	Požadovaná minimální délka zadávaného textu
maxLength	Požadovaná maximální délka zadávaného textu
minValue	Minimální číselná hodnota
maxValue	Maximální číselná hodnota
defaultValue	Možnost nastavit výchozí hodnotu
label	Popisek který se má zobrazit, mohou mít všechny komponenty

[31] Grewe J, Wachtler T and Benda J (2011) A bottom-up approach to data annotation in neurophysiology. *Front. Neuroinform.* **5**:16. doi: 10.3389/fninf.2011.00016

Combobox

Combobox poskytuje rychlý způsob jak vybrat jednu hodnotu z množiny hodnot. Definují se pomocí *Vlastnosti.* Příklad na následující ukázce:

```
<section>
      <type>combobox</type>
      <name>pohlaví</name>
      <property>
          <name>datatype</name>
          <value>string</value>
      </property>
      <property>
          <name>values</name>
          <value>Muž</value>
          <value>Žena</value>
      </property>
  </section>
```

Checkbox je zaškrtávací tlačítko, které má dva stavy zaškrtnuto/nezaškrtnuto.

Choice Podobné komponentě Combobox s tím rozdílem, že umožňuje vybrat z většího množství hodnot.

4.1.3 Určení pozice

Abychom mohli určit pozici jednotlivých polí ve formuláři, musí jednotlivé *Sekce* obsahovat *Vlastnosti* uvedené v tabulce 3.

Tabulce 3 - Poziční identifikátory[32]

VLASNOST	POPIS
id	Unikátní identifikátor komponenty
idTop	ID komponenty umístěné nad stávající
idLeft	ID komponenty umístěné nalevo od stávající

[32] Vlastní analýza

5 Analýza požadavků mobilní aplikace

Cílem kapitoly je navrhnout specifikaci požadavků pro novou mobilní aplikaci.

5.1 Základní objekty pro tvorbu formulářů

Mezi základní objekty pro tvorbu formulářů řadíme formulář, pole a šablonu uživatelského rozhraní.

Formulář je základní komponenta uživatelského rozhraní. Skládá se z jednotlivých polí.

Pole slouží k vyplnění konkrétních informací. K poli lze zadat titulek (Label). Mohou se vyskytovat následující typy polí:

Textbox – textové pole, které může nabývat datového typu String, Integer, Number nebo Date.

Form – na formuláři může být umístěn libovolný podformulář. Například formulář Experiment může obsahovat podformulář se seznamem osob účastnících se daného experimentu.

Checkbox (zaškrtávací pole) - Checkbox je zaškrtávací tlačítko, které má dva stavy zaškrtnuto/nezaškrtnuto.

Combobox (výběrové pole) - Combobox poskytuje rychlý způsob jak vybrat jednu hodnotu z množiny hodnot.

Šablona uživatelského rozhraní - Uživatel si může libovolně navolit rozložení polí v oblasti formuláře dle aktuálních potřeb. Výsledek rozložení si pak uloží do své šablony uživatelského rozhraní (Layout). Pokud nepotřebuje tvořit šablonu novou, tak může použít již vytvořené šablony jinými uživateli.

5.2 Funkce mobilního klienta

Při vytváření pracovního prostoru (workspace) vyplní uživatel jeho název a může nastavit přihlašovací údaje a adresu serveru se kterým bude chtít provádět synchronizaci dat. Údaje o serveru nejsou povinné, poněvadž aplikace musí umožnit

funkcionalitu i bez připojení k serveru. Výhodou je, že uživatel může být pomocí pracovních prostorů připojen k různým serverům.

Pokud uživatel vyplní přihlašovací údaje k serveru, tak se mu zobrazí seznam všech dostupných layoutů. Ze zobrazené nabídky layoutů si může zvolit, které chce přidat do pracovního prostoru.

Přidání formuláře do pracovního prostoru

Zde jsou dvě možnosti, buď se zobrazí seznam již existujících layoutů nebo si uživatel může vytvořit formulář zcela nový, který odpovídá potřebám.

Vytvoření formuláře

Pokud se uživatel rozhodne tvořit vlastní formulář, je třeba nejprve zadat typ formuláře. Nyní přistoupí ke druhému kroku, kterým je definice jednotlivých polí ve formuláři.

Vytváření záznamu do formuláře

Klient si do existujícího formuláře zaznamenává údaje (například naměřená data z experimentu).

Editace formuláře

Během editace formuláře může uživatel přidávat, editovat či mazat pole ve formuláři. Libovolně též může měnit rozložení polí ve formuláři.

Synchronizace dat

Je-li mobilní telefon připojen k Internetu a uživatel zadal server, se kterým chce provést synchronizaci dat, tak aplikace umožní synchronizovat definici polí, formuláře, layouty a vyplněná data formulářů. Pokud mobilní telefon připojen k internetu není, synchronizace neproběhne.

5.3 Databáze

Vytvořené formuláře včetně dat vyplněných polí lze uložit do relační databáze na daném mobilním zařízení. Je třeba navrhnout vhodnou databázovou strukturu.

5.4 Offline práce s daty

Během experimentů jsou shromažďována metadata. Systémy pro shromažďování metadat poskytují uživateli webové rozhraní, prostřednictvím kterého je umožněno uživateli vkládat informace. Problém nastává v situaci, kdy se uživatel nachází v prostředí, kde není možné připojit se k internetu (například nemocnice). V dnešní době už jsou "chytré telefony" natolik rozšířené, že nám tento problém mohou pomoci vyřešit. Řešením je klientská aplikace, která pracuje v režimu offline. V okamžiku sběru dat, data uloží. Uživatel provede synchronizaci později, když je mobilní zařízení připojeno k Internetu.

6 Mobilní klient pro elektrofyziologické databáze

Mnou vytvářená mobilní aplikace by měla usnadnit práci uživatelům při práci s naměřenými daty z neurovýzkumu.

6.1 Struktura aplikace

Minimální verze SDK, na které lze aplikaci zprovoznit je verze 14 (Android 4.0 ice cream sandwich). Cílovou verzí je verze 19 (Android 4.2.2). Hlavním důvodem je klesající zastoupení předchozích verzí a problém částečně odlišného vývoje při používání podpůrných knihoven, pokud chceme používat funkce z novějších API (Fragmenty, ActionBar). Dalším důvodem byla nemožnost použití Drag and Drop na objekty uživatelského rozhraní (přidáno ve verzi 14), což je jedna ze zásadních a podstatných vlastností editoru formulářů naší aplikace.

Verze SDK má též vliv na vzhled aplikace jako takové. Aplikace nevyžaduje jiná specifická omezení snad jen to, že potřebuje mít povolen přístup k internetu a umožněn číst stav sítě. Tyto informace jsou uvedeny v `AndroidManifest.xml`, kde jsou též popsány veškeré aktivity z balíku `ui`.

Aplikace je rozdělena do několika funkčních celků. Manipulace s daty se nachází v balíku `data`. V průběhu vývoje byl kladen důraz, abychom mohli datovou část při minimálních změnách (s drobnými úpravami) použít i v jiné aplikaci.

6.2 Data aplikace

Mobilní aplikace pracuje s daty formulářů. V této kapitole popíši, jakým způsobem jsou data získávána, zpracována a ukládána.

6.2.1 Získávání dat

Webové služby

Data formulářů, mezi které patří schémata formulářů a jeho data, jsou získávána pomocí webových služeb. Umožňují jednoduchou komunikaci mezi aplikacemi v heterogenních prostředích. Poskytovány jsou dva základní standardy webových

služeb: SOAP (Simple Object Access Protocol) a REST (Representation State Transfer)[33]. Protokol SOAP je spíše zaměřen na přístup k operacím vzdáleného systému, REST pohlíží na data takového systému.

Rozhodli jsme se pro použití REST služeb, poněvadž potřebuji získávat jen samotná data. Výhodou REST je snaha o maximální jednoduchost při přistupování k datům. Rest je architektura, která umožňuje přistupovat k datům na určitém místě pomocí standardních metod http protokolu, na které jsou mapovány základní CRUD[34] metody.

Získávání dat v aplikaci - Rest Template[35]

Projekt Spring pro Android[36] poskytuje funkce pro vytváření klientské aplikace založené na REST. Implementace se nachází v balíku `ws.` Použití je velmi jednoduché. Základem pro použití je vytvořit instanci třídy `RestTemplate,` sloužící jako abstrakce mezi jednotlivými http požadavky. RestTemplate používá nativní knihovny pro vytváření http požadavků. Velkou výhodou je, že s různými verzemi Android SDK jsou i odlišné implementace těchto knihoven a použitím RestTemplate programátor nemusí obsluhovat každou zvlášť. RestTemplate podporuje různé formáty pro čtení a zápis. Výstupem použité odML knihovny je XML. Abychom zprávy mohli zabalit do XML formátu, je třeba nastavit SimpleXmlHttpMessageConverter.

Vytvořením instance třídy HttpHeaders můžeme definovat HTTP hlavičku. V aplikaci se nastavuji dva parametry:

---

[33] Stag: rest-architektura-pro-webove-api. *Rest-architektura-pro-webove-api* [online]. [cit. 2014-06-20]. Dostupné z: https://stag-ws.tul.cz/ws/help?page=tech%20http://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

[34] CRUD - Create, Read, Delete, Update

[35] DAHANNE, Anthony. *Instant Spring for Android Starter*. USA: Packt Publishing, 2013. ISBN 1782161902.

[36] Spring for Android Reference Documentation. *Spring.io* [online]. 2014 [cit. 2014-06-20]. Dostupné z:http://docs.spring.io/spring-android/docs/1.0.2.BUILD-SNAPSHOT/reference/htmlsingle/

1. Authorization: Klient se musí ověřit pomocí uživatelského jména a hesla, dle standardního mechanismu HTTP BASIC. Realizace je vytvoření objektu HttpAuthentication, který předáme objektu třídy `HttpHeaders`.
2. Accept: Definujeme typ, jaký je přijatelný pro odpověď našeho požadavku. Nastavením APPLICATION_XML říkáme, že požadujeme XML formát.

Webové služby voláme přes zabezpečenou vrstvu HTTPS, kde data jsou šifrována pomocí SSL. Implementaci jsem převzal od Petra Mika[37]. Aplikace potvrzuje veškeré certifikáty použité při komunikaci. Účelem je vytvoření funkčního prototypu, kde komunikace nemá oficiální certifikát. Toto řešení by mělo být pouze dočasné. Při realizaci komunikace je v knihovně Spring chyba. Standardní komunikace v RestTemple je napřed bufferována, což způsobuje problémy s pamětí při odesílání velkého množství dat. Petr Miko tento problém ve své implementaci vyřešil.

Výsledkem je vytvoření třídy `SSLSimpleClientHttpRequestFactory`, která je oddělena ze třídy `SimpleClientHttpRequestFactory`, jenž je součástí Spring frameworku.

Posledním krokem je vzájemná výměna dat mezi klientem a serverem. Provádí se zavoláním funkce `exchange`. Jako parametr se nastavuje: adresa služby http metody, kterou budeme používat, entita s nastavením hlaviček a nakonec formát příchozích zpráv. Ukázka takové komunikace je níže:

```
HttpAuthentication authHeader = new
HttpBasicAuthentication(user,password);
HttpHeaders requestHeaders = new HttpHeaders();
requestHeaders.setAuthorization(authHeader);
requestHeaders.setAccept(Collections.singletonList(MediaType.APPLICATI
ON_XML));
HttpEntity<Object> entity = new HttpEntity<Object>(requestHeaders);
SSLSimpleClientHttpRequestFactory factory = new
SSLSimpleClientHttpRequestFactory();

RestTemplate restTemplate = new RestTemplate(factory);
restTemplate.getMessageConverters().add(
```

[37] MIKO, Petr. *Mobile system for management of EEG/ERP experiments*. Plzeň, 2013. Master Thesis. University of West Bohemia.

```
    new SimpleXmlHttpMessageConverter());

ResponseEntity<LayoutList> responseList = restTemplate.exchange(
    urlAvailableLayouts, HttpMethod.GET, entity, LayoutList.class);
LayoutList body = responseList.getBody();
FormBuilder formBuilder = new FormBuilder(fragment.getDaoFactory(),
result);
```

Přijatá data v odML formátu (data o formuláři), jsou do databáze ukládány třídou `FormBuilder`, kterou popíši níže. Ostatní data ve formátu XML (např. seznam layoutů na serveru, informace o přihlášeném uživateli) konvertuji na Java objekty pomocí technologie JAXB (Java Architecture for XML Binding). Realizaci provádím anotováním tříd POJO (Plain Old Java Object).

Ukládání a přístup k datům[38]

Veškerá data o formulářích a pomocných strukturách aplikace jsou ukládána v Android databázi SQLite. Jsou ukládána do databáze, abychom uživateli umožnili pracovat s aplikací bez přístupu k internetu. Aktualizace dat probíhá, pouze pokud k tomu uživatel vyzve. SQLlite reprezentuje relační databázový systém, který je obsažen v jedné malé knihovně. Výhodou SQLite je rychlost a malá paměťová náročnost, což jej předurčuje k použití na mobilních zařízení. Pro prohlížení databáze přímo v Eclipse, je třeba mít nainstalovaný nástroj *Questoid SQLite Browser,* který se integruje do Dalvik Debug Monitor Server (DDMS). Prohlížení databáze funguje pouze při spuštění virtuálního stroje nebo na telefonu s administrátorským oprávněním.

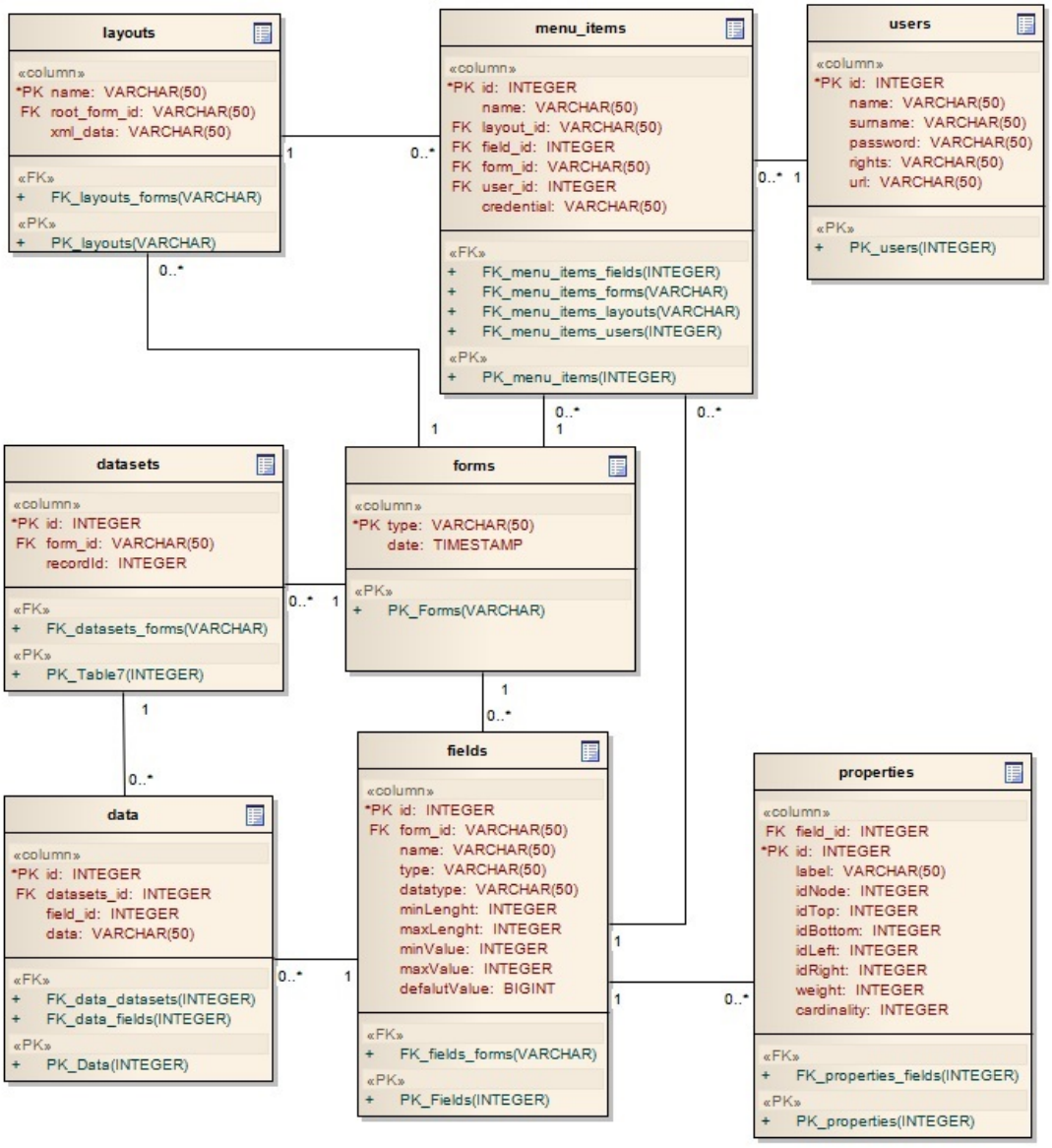
Databáze je uložena v samostatném souboru eegMobileDatabase.db. Umístěna je v *data/data/cz.zcu.portal.android/databases/*. K vytvoření Databáze je třeba vytvořit vlastní třídu, od které oddělíme třídu `SQLiteOpenHelper` a přepíšeme metody

[38] VÁVRŮ, Jiří a Miroslav UJBÁNYAI. Programujeme pro Android. Česká Republika: Grada, 2013. ISBN 978-80-247-4863-4.

onUpgrade() a onCreate(). Pro získávání dat slouží metoda query nebo pro složitější dotazy SQLiteQueryBuilder.

Na obrázku 8 je ER diagram databáze. Tabulka layouts slouží k ukládání šablon uživatelského rozhraní. Šablona obsahuje typ formuláře, ke kterému se vztahuje. Jeden formulář může mít definováno několik šablon. Tímto způsobem si každý uživatel může vytvořit svoje šablony uživatelského rozhraní podle svých vlastních preferencí. Formulář může obsahovat libovolné množství polí, které ukládám do tabulky fields. Tabulka dataset představuje záznam vyplněného formuláře, seskupuje vyplněná data polí do jednoho záznamu. Mezi tabulkou fields a tabulkou layouts je tabulka properties, ve které se nacházejí rozšiřující vlastnosti pole umístěného na layoutu (například pozice pole na layotu). Tabulka menu_items slouží k ukládání složek pracovních prostorů. V tabulce users jsou vyplněné přihlašovací údaje a adresa serveru se kterým komunikujeme.

Obrázek 8 - ER diagram[39]



[39] Zdroj: vlastní analýza.

Objektově relační mapování (ORM)[40]

Pro usnadnění práce s daty, bylo využito objektově relační mapování. Zvolil jsem knihovnu OrmLite, která umožnuje jednoduchou perzistenci Java objektů v SQLite databázi.

Anotacemi jsou označovány třídy, které mají sloužit k perzistenci. Třída označená anotací @DatabaseTable odpovídá tabulce, ve které je objekt uložen. Atributy třídy, které chceme mít v tabulce, označíme anotací @DatabaseField.

Na následující ukázce je třída Field obsahující data jedné položky na formuláři. Odpovídající tabulka se jmenuje fields. ORMlite nepodporuje složený primární klíč z několika sloupců. V tomto případě je třeba, aby formulář obsahoval pouze jedno pole daného jména. Řešením je vytvoření unikátního indexu uniqueIndexName.

```
HttpAuthentication authHeader = new
HttpBasicAuthentication(user,password);
@DatabaseTable(tableName = Field.TABLE NAME)
public class Field {

        public static final String TABLE_NAME = "fields";
        public static final String FK_ID_FORM = "form_id";
        public static final String FIELD_ID = "field_id";
        public static final String DATA_TYPE = "data_type";
        public static final String INDEX_NAME = "field_form_name_idx";

        @DatabaseField(generatedId = true)
        private int id;

        @DatabaseField(uniqueIndexName = INDEX_NAME)
        private String name;

        @DatabaseField
        private String type;

        @DatabaseField(columnName = DATA_TYPE)
        private String dataType;

        @DatabaseField(foreign = true, canBeNull = false, columnName =
            FK_ID_FORM, uniqueIndexName = INDEX_NAME)
        private Form form;

        ...
}
```

[40] SESSA, Carlos. *50 Android Hacks*. USA: Manning Publications, 2013. ISBN 978-1617290565.

V databázi je mezi tabulkou Layout a Field vazba M:N, proto bylo nutné vytvořit propojující tabulku properties. Ta obsahuje primární klíč ID, jenž je automaticky generován. Cizí klíče jsou objekty třídy `Field` a `Layout`. Podrobněji situaci vidíme v ukázkové anotaci:

```
@DatabaseTable(tableName = LayoutProperty.TABLE_NAME)
public class LayoutProperty {

public static final String TABLE_NAME = "properties";
        public static final String FK_ID_FIELD = "field_id";
        public static final String FK_ID_LAYOUT = "layout_id";

        @DatabaseField(generatedId = true)
        private int id;

        @DatabaseField(foreign = true, canBeNull = false, columnName =
          FK_ID_FIELD)
        private Field field;

        @DatabaseField(foreign = true, canBeNull = false, columnName =
          FK_ID_LAYOUT)
        private Layout layout;
```

Pro přístup a správu databáze slouží třída `DatabaseHelper`, která dědí od třídy `OrmLiteSqliteOpenHelper`. Tato třída umožňuje vytvářet a aktualizovat databázi, když je aplikace nainstalována. Z tohoto důvodu musíme implementovat metody onCreate a onUpgrade. Metoda onUpgrade je volaná při aktualizaci aplikace a je potřeba provést změnu struktury databáze. Třída `DatabaseHelper` poskytuje přístup k třídám *`Data Acces Object`* `(DAO)`, objekty DAO poskytují základní CRUD operace. DAO třídy, rozšiřují funkcionalitu tříd základních. Nacházejí se v balíku data.dao a obsahují veškeré dotazy do databáze. Přistup k vytvořeným třídám je zajištěn pomocí tovární třídy DAOFactory, která vytváří instance k jednotlivým DAO třídám a při vytváření jí předávám odkaz na kontext aplikace. DAOFactory přistupuje ke třídě `OpenHelperManager`. Pokud totiž otevíráme více spojení ke stejné databázi, tak může dojít k získání nekonzistentních dat, proto je doporučeno používat `OpenHelperManager` ke sledování stavu databáze. Na následující ukázce je získání šablony daného jména.

```
String name;
...
if (layout == null) {
  layout = daoFactory.getLayoutDAO().getLayout(name);
}
```

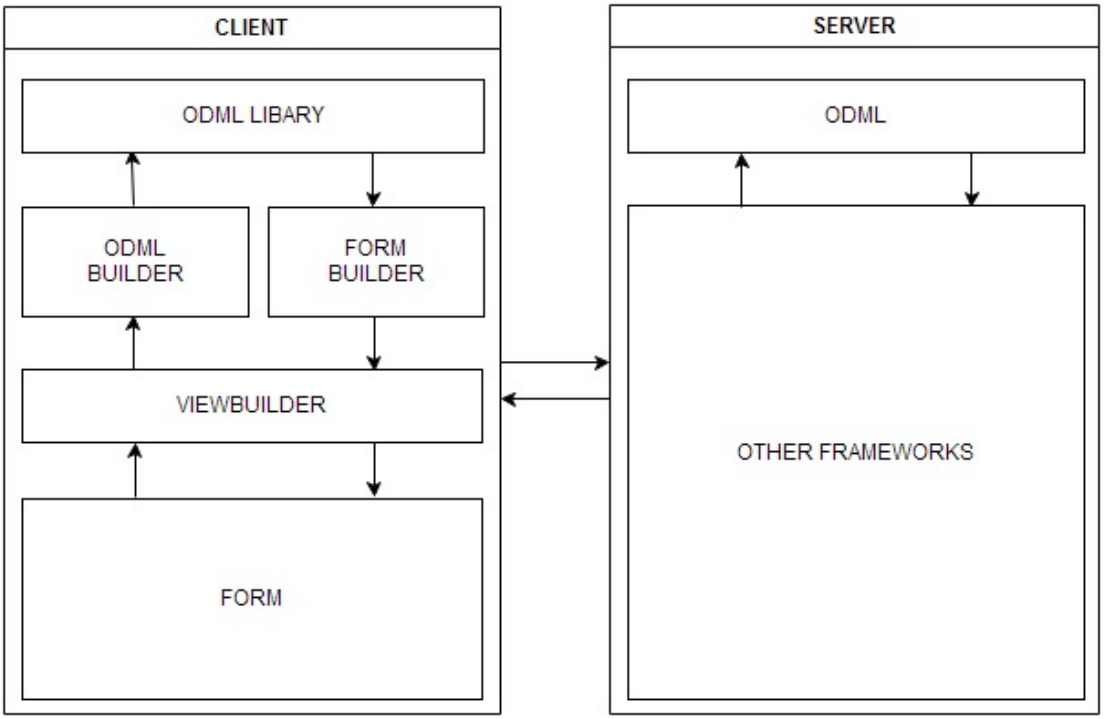
Pro tvorbu dotazů do databáze využívám v DAO třídách `QueryBuilder` umožňující zápis složitějších dotazů postupným zřetězováním metod. Na následujícím výpisu je ukázka, jak získat záznam z tabulky properties na základě pole a layoutu.

```
QueryBuilder<LayoutProperty, Integer> queryBuilder =
   getLayoutPropertyDao().queryBuilder();
Where<LayoutProperty, Integer> where = queryBuilder.where();
where.eq(LayoutProperty.FK_ID_FIELD, fieldId);
where.and();
where.eq(LayoutProperty.FK_ID_LAYOUT, layoutId);
PreparedQuery<LayoutProperty> preparedQuery = queryBuilder.prepare();
return getLayoutPropertyDao().queryForFirst(preparedQuery);)
```

Zpracování dat

V balíku `data.builders` se nacházejí čtyři velmi důležité části systému. Architektura je uvedena na obrázku 9. Aplikace komunikuje se serverem pomocí webových služeb. Přijatou šablonu je nejprve potřeba načíst z formátu odML a uložit do databáze na mobilním zařízení. K tomuto účelu slouží třída `FormBuilder`, která pro tento účel volá funkce z knihovny oDML. Třída `FormBuilder` získává údaje o šabloně, jako je jméno šablony, typ formuláře a následně rekurzivně prochází stromovou strukturu *Sekcí,* ze které zjistí jednotlivá pole a jejich vlastnosti a vše se uloží do databáze.

Obrázek 9 - Základní architektura[41]

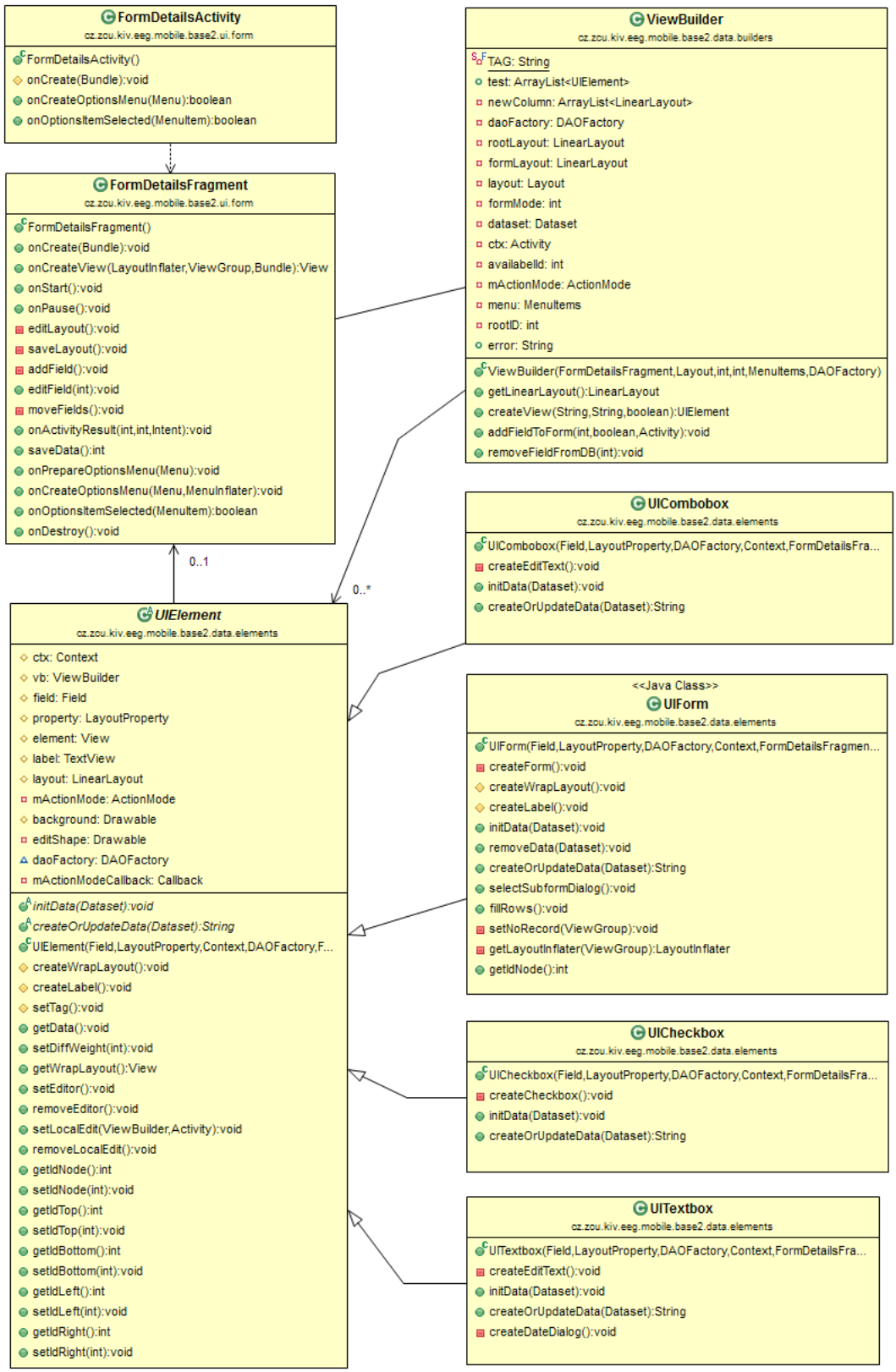


Základní třídou pro generování uživatelského rozhraní je ViewBuilder. Zde převádí námi obecně definované pole v odML terminologii na grafické objekty Androidu (View), např. Textbox je v Androidu definován objektem EditText, Combobox pomocí objektu Spinner. Dále se zde registrují posluchače událostí daných objektů např. pro Drag and Drop. Více uvedu v kapitole kapitole 6.4.

Uživatel aplikace si může upravit rozložení formuláře nebo si vytvářet svoje vlastní formuláře. Rozložení je třeba převést opačným způsobem a to tedy z Androidu do odML formátu. Třída `ViewBuilder` (obrázek 10) prochází objekty uživatelského rozhraní (`UIElement`) a zjistí, kde se ve formuláři nacházejí. Výslednou strukturu předá třídě `OdmlBuilder`, která se postará o převedení do odML formátu.

[41] Zdroj: Vlastní analýza

Obrázek 10 – UML diagram tříd na tvorbu šablony



6.3 Paralelní vykonávání kódů [42]

Aktivity (Activity)[43] prezentují prezenční vrstvu. Jedná se o základní vizuální komponentu, reprezentující jednu obrazovku aplikace, sloužící k interakci s uživatelem. Pokud potřebujeme realizovat časově náročnou operaci, je vhodné výpočet přesunout do paralelního vlákna jinak riskujeme, že právě probíhající Aktivita nebude schopna správně vykreslovat uživatelské rozhraní nebo obsluhovat uživatelské vstupy.

Při spuštění aplikace se vytvoří nový linuxový proces s jedním vláknem (Main Thread), často je nazýván jako UI thread, protože aktualizuje uživatelské rozhraní (UI - user interfrace). Android nepodporuje aktualizaci uživatelského rozhraní z ostatních vláken, podporuje aktualizaci uživatelského rozhraní pouze z hlavního vlákna.

Dalším důvodem pro paralelní vykonávání, jsou síťové operace, které mohou trvat dlouhou dobu. Android zakázal síťové operace od verze 3.0 (Honeycomb), pokud nejsou vykonávány v samostatném vlákně. Pokud bychom nedodrželi implementaci síťových operací do samostatného vlákna, nastala by výjimka *NetworkOnMainThreadException*. Android poskytuje hned několik možností jak pracovat s vlákny.

6.3.1 AsyncTask

AsyncTask nabízí jednoduchý způsob jak manipulovat s UI vláknem. Tato třída poskytuje operace k provádění operací na pozadí a výsledek umožní předat UI vláknu. AsyncTask používám u implementace webových služeb v balíku ws. Třídy, které používají AsyncTaks musí dědit od Třídy AsyncTask<Params, Progress, Result>, která obsahuje tři generické typy:

1. Params – typ vstupních parametrů potřebných k výpočtu
2. Progress – typ parametrů při aktualizaci průběhů výpočtu
3. Result – typ výsledků výpočtu

---

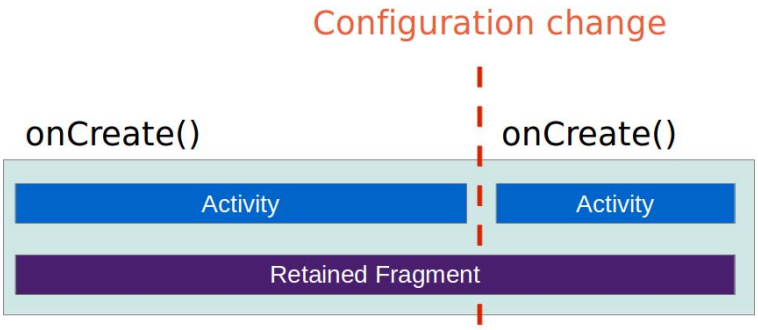
[42] PHILLIPS, Bill. *Android Programming: The Big Nerd Ranch Guide (Big Nerd Ranch Guides)*. USA: Big Nerd Ranch Guides, 2013. ISBN 978-0321804334.

Funkce onPreExecute() se volá před samotným výpočtem. Zde aktualizuji dialogy informující o stavu síťové komunikace se serverem. Výpočet v novém vlákně probíhá v metodě doInBackground(Params...), kde se nachází komunikace se serverem a výměna dat. Aktualizaci progress dialogů provádím v metodě onProgressUpdate(Progress...). Předávání výsledných dat Aktivitě probíhá v metodě onPostExecute(Result). V této metodě máme vyvolané UI vlákno, které převezme a zpracuje výsledky.

Použití je velmi jednoduché, problém vzniká v případě, že dojde k ukončení Aktivity, na kterou má vlákno referenci. Ukončení Aktivity proběhne například při změně orientace obrazovky. Pokud bych tento problém neošetřil, dojde k pádu celé aplikace. Vyřešil jsem to použitím Fragmentu.

Fragmenty slouží podobně jako Aktivity k definování uživatelského rozhraní. Vznikli z důvodu vytvoření uživatelského rozhraní na tablety. Fragmenty obsahují části uživatelského rozhraní s vlastní logikou. Jejich životní cyklus je řízen Aktivitou, ve které se nachází. Je možné vytvořit tzv. headless fragmenty, které dokáží uložit data mezi změnou konfigurace. Jejich životní cyklus je po dobu činnosti Aktivity (obrázek 11). Headless Fragmenty nemají definované uživatelské rozhraní, pouze slouží k uchovávání stavu a proto jsou označovány jako headless.

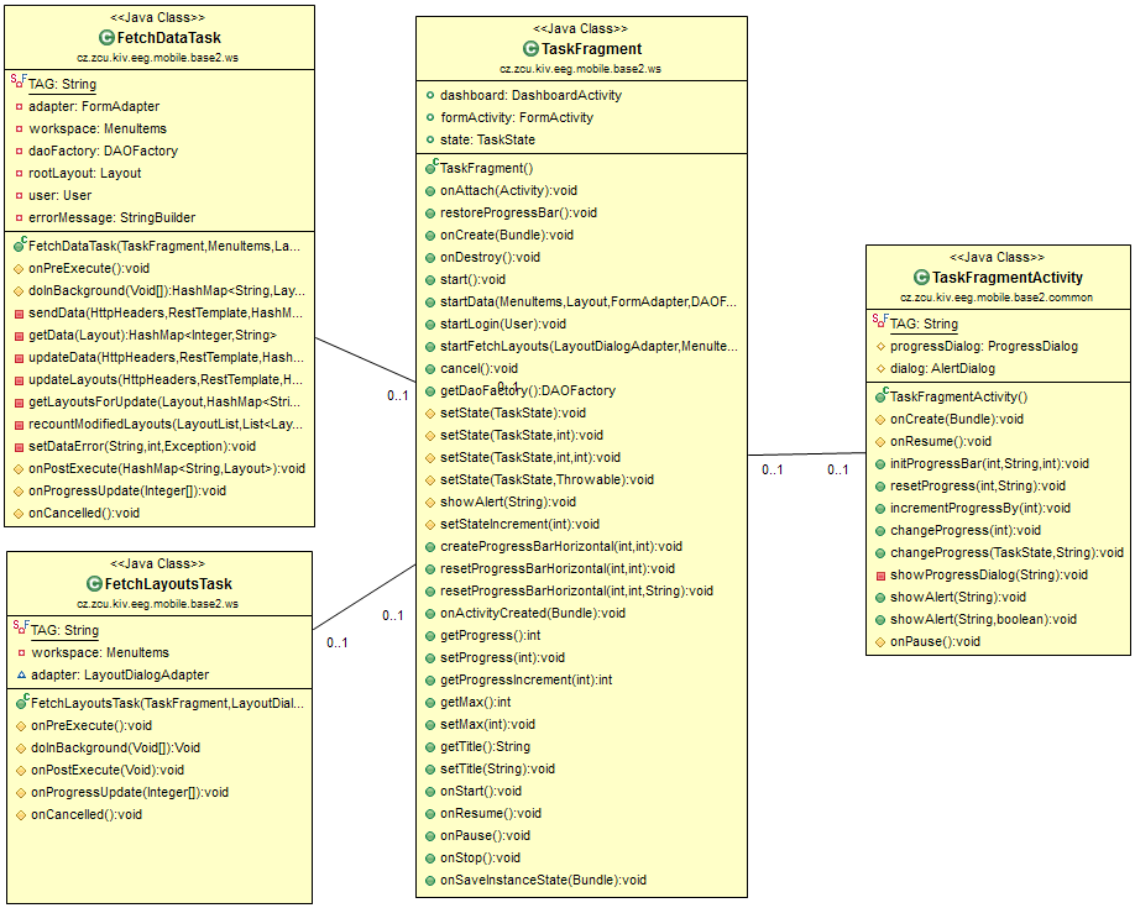
Obrázek 11 - Headless Fragment[44]



[44] Www.vogella.com: Multi-pane development in Android with Fragments - Tutorial. *Vogella* [online]. 2013 [cit. 2014-06-21]. Dostupné z: http://www.vogella.com/tutorials/AndroidFragments/article.html

Vytvořil jsem třídu `TaskFragment` (obrázek 12), která dědí od třídy `Fragment`. Třída slouží k uchovávání reference na Aktivitu `TaskFragmentActivity` i při změně stavu. Abych toho docílil `TaskFragment` nesmí obsahovat uživatelské rozhraní a v metodě onCreate() volám metodu setRetainInstance(). V této třídě vytvářím konkrétní instance třídy `AsyncTask`, které slouží ke komunikaci se serverem. `TaskFragment` vlastně obaluje jednotlivé instance třídy `AsyncTask` a tímto způsobem je zajištěna existence reference na Aktivitu i při změně konfigurace.

Obrázek 12 – UML diagram asynchronního stahování dat



Ve třídě `TaskFragment` se nacházejí metody pro obsluhu chybových zpráv. Pokud dojde k chybě při komunikaci se serverem, chyba je předána do `TaskFragmentu` a jelikož on jako jediný má platnou referenci na Aktivitu, chybovou zprávu předá Aktivitě k zobrazení. Následuje ukázka vytvoření `TaskFragmentu`

v Aktivitě, který vytvoří instanci třídy `FetchLayoutsTask`, jenž dědí od `AsycnTask` a umožní nám v novém vlákně stáhnout seznam šablon:

```
TaskFragment mTaskFragment;

FragmentManager fm = getFragmentManager();
mTaskFragment = (TaskFragment) fm.findFragmentByTag(TAG + "Fragment");
if (mTaskFragment == null) {
    mTaskFragment = new TaskFragment();
    fm.beginTransaction().add(mTaskFragment,"taskFragment").commit();
}

mTaskFragment.startFetchLayouts();
```

6.4 Vzhled a ovládání aplikace

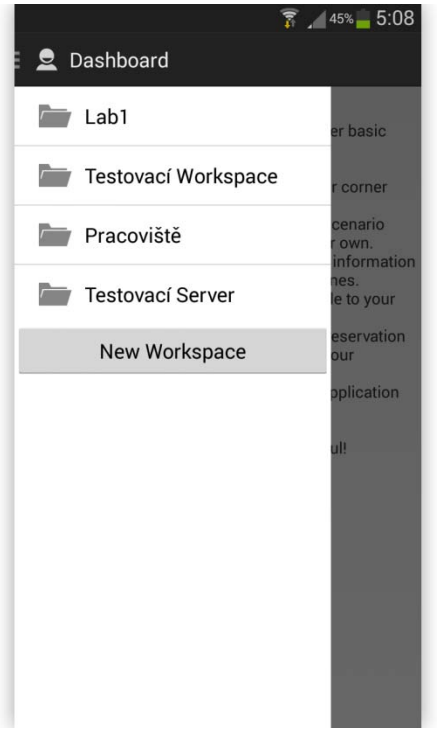
Třídy související se vzhledem a zobrazením dat z formulářů se nacházejí v balíčku `cz.zcu.kiv.eeg.mobile.base2.ui`. Nacházejí se zde pouze Aktivity tvořící uživatelské rozhraní. Některé části statických obrazovek jsou popsány pomocí XML šablon (např. položky seznamů, dialogová okna, Menu), které se nacházejí v *res/layout*. Ostatní obrazovky hlavně tvorba formulářů se vytváří přímo ve zdrojovém kódu. Jedná se dynamické obrazovky. Následně budou popsány základní prvky uživatelského rozhraní.

Výsuvný panel Drawer

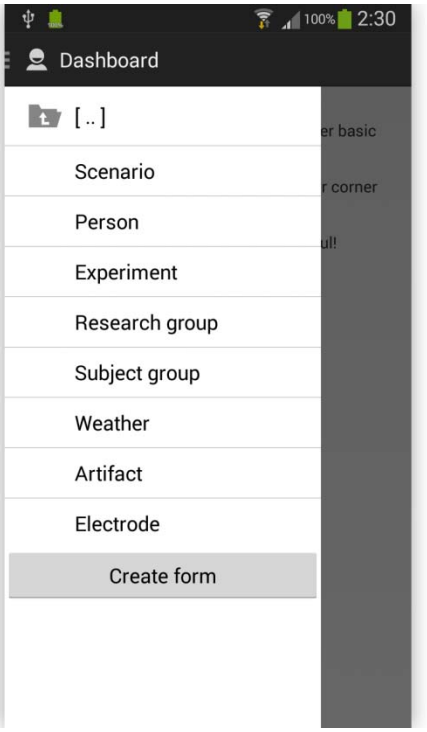
K zobrazení seznamu dostupných pracovních prostorů slouží navigační panel (obrázek 13), který je většinu času schován. Zobrazí se přejetím prstu z levého okraje obrazovky nebo kliknutím na ikonku Dashboard. Panel je vytvořen ve třídě `DashboardActivity`. Jako první je nutno inicializovat seznam položek, které jsou zobrazovány v ListView. Dále je vytvořen adaptér, který poskytuje přístup k datovým položkám a zároveň je zodpovědný za tvorbu vzhledu. Seznam položek je načítán z databáze a předán do DrawerAdapter, který je zděděn od ArrayAdapter<MenuItems>. Jakmile uživatel klikne na položku, tak systém zavolá metodu onItemClick(), ve které je panel překreslen seznamem dostupných layoutů. Při vybrání layoutu se otevře nová Aktivita.

Při práci s panelem jsem si všiml chyby. Pokud otevírám novou Aktivitu a současně zavřu panel. Aktivita se nebude otevírat plynule. Chyba se projevuje na různých zařízení a u všech aplikací.

Obrázek 13 - Navigační panel

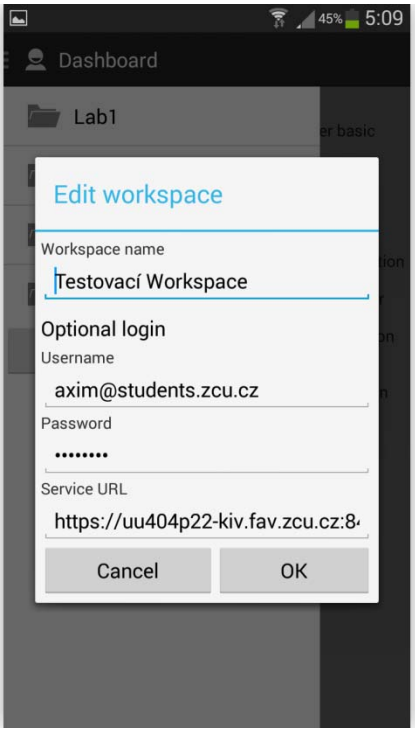


Obrázek 14 – Seznam šablon

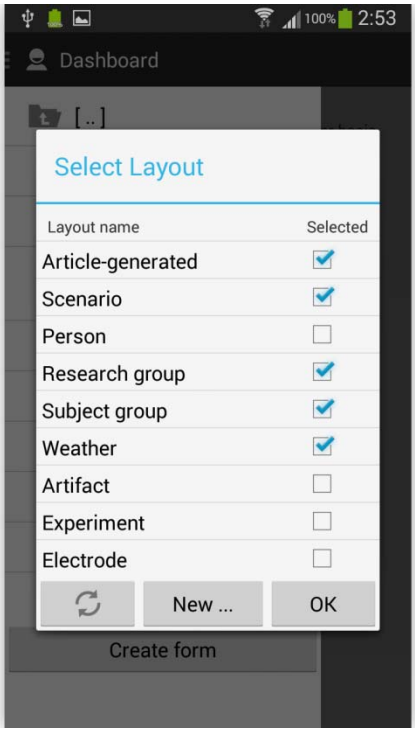


Kliknutím na složku pracovního prostoru se načte seznam šablon jednotlivých formulářů (obrázek 14). Dlouhým kliknutím na libovolné položky v seznamech lze vyvolat kontextové menu, které nám umožní editovat zvolenou položku. Na obrázku 15 je vyvolaná editace pracovního prostoru pomocí třídy Dialog, které nastavím zdroj uživatelského rozhraní definované v šabloně workspace_add.xml. Třída `Dialog` slouží k návrhu vlastních dialogů. Pro tvorbu dialogu chybových zpráv slouží třída `AlertDialog`, která má předpřipravené tří volitelná tlačítka. Přidat nový pracovní prostor lze tlačítkem *New Worskpace,* kterým se otevře stejný dialog jako při editaci pracovního prostoru na obrázku 15. Změna je pouze v titulku dialogu.

Obrázek 15 – Editace pracovního prostoru



Obrázek 16 – Výběr šablon

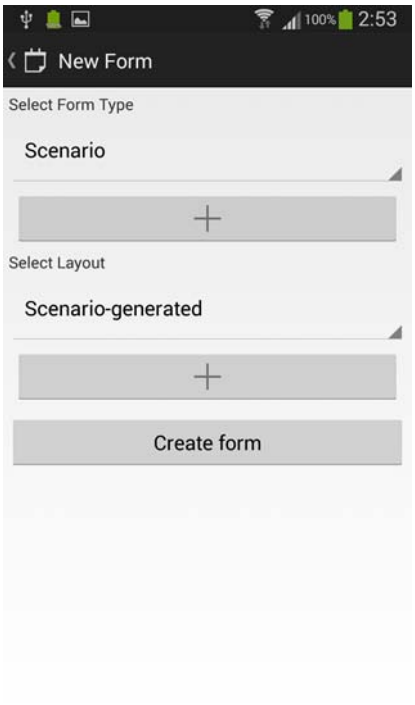


Při vytváření pracovního prostoru je povinné vyplnit jméno pracovního prostoru. Uživatelské jméno, heslo a adresa serveru jsou volitelné položky. Jakmile je jedna z nich vyplněná, tak při stisknutí tlačítka *OK* se data odešlou na server a provede se jejich kontrola. Pokud je vše v pořádku vytvoří se nový pracovní prostor a otevře se dialog na přidání nových šablon (obrázek 16). Pokud uživatel vyplnil přihlašovací údaje k serveru, je zobrazeno navíc tlačítko pro stažení šablon ze serveru. Na stahování šablon se zavolá AsyncTask ze třídy FetchLayoutsTask. Průběh stahování se zobrazuje pomocí ukazatele ze třídy ProgressDialog. Jakmile je stažení šablon dokončeno, aktualizuje se seznam dostupných šablon. Při velkém počtu šablon docházelo při posouvání seznamu šablon k odškrtnutí vybraných položek, pokud se dostaly mimo viditelnou část obrazovky. V adaptéru `LayoutDialogAdapter` jsem musel vytvořit třídu `ViewHolder`, která obsahuje atribut jméno layoutu a checkbox. Hlavní trik spočívá v inicializaci checkboxu v adaptéru. Při inicializaci řádky adaptéru ve funkci initView se vytvoří instance třídy `ViewHolder` a pro atribut checkbox se nastaví Tag na pozici řádky. Při změně stavu checkboxu se zavolá metoda onCheckedChanged, kde se

nalezne pozice pomocí hodnoty uložené v Tag a ze seznamu hodnot se vybere správná řádka, které se nastaví změněná hodnota.

Vytvořit nový formulář nebo šablonu lze tlačítkem „*New…*“. Posléze dojde k otevření Ativity `FormAddActivity` na obrázku 17. Na této obrazovce lze kliknutím na tlačítko + vytvořit nový formulář a šablonu nebo vybrat existující formulář a k němu vytvořit novou šablonu. Při zvolení existujícího formuláře se automaticky načtou jeho šablony v poli *Select Layout.* Toho bylo docíleno přidáním posluchače setOnItemSelectedListener na pole *Select Form Type.* Pokud je vytvářen nový formulář automaticky se vytvoří šablona obsahující jediné pole *description.* Kliknutím na tlačítko *Create Form* se šablona přidá do zvoleného pracovního prostoru a okno se uzavře a zobrazí se seznam šablon (obrázek 14).

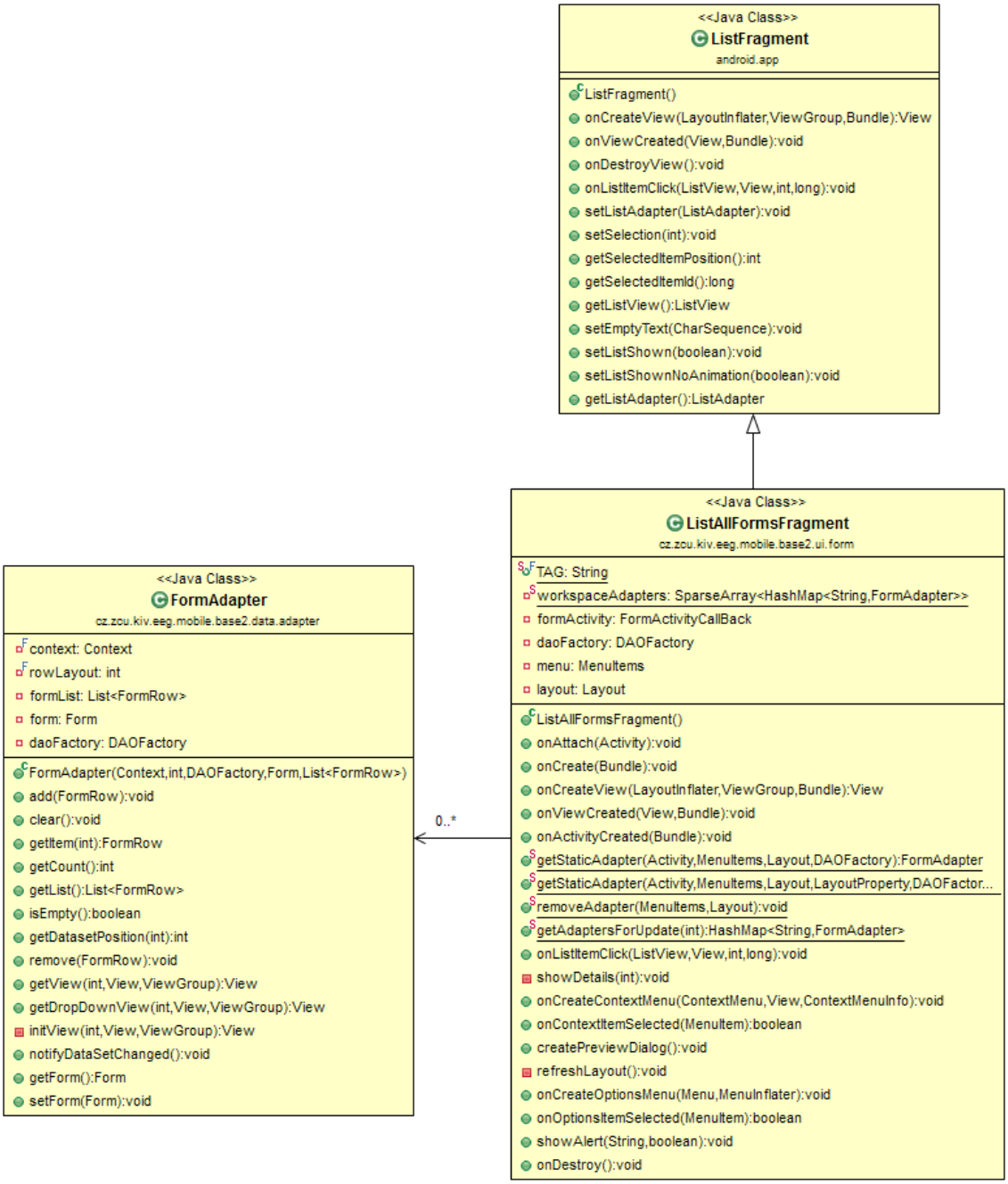
Obrázek 17 – Vytvoření nového formuláře



Zobrazení dat

Kliknutím na jméno šablony nacházející se v pracovním prostoru (obrázek 14) se zobrazí veškeré záznamy daného formuláře. Pro formulář Person je to seznam jmen (obrázek 19). O zobrazení se stará Aktivita `ListAllFormsFragment`. Poněvadž je tato třída zděděná od `ListFragment`, tak se musí jednotlivé záznamy ukládat do adaptéru třídy `FormAdapter` (obrázek 18).

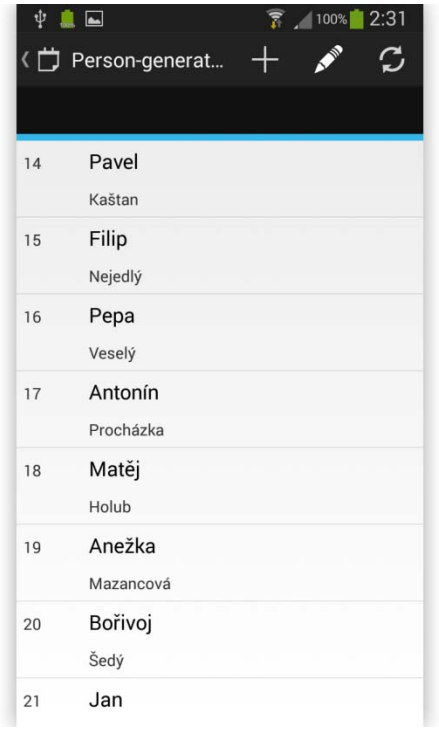
Obrázek 18 – UML diagram seznamu šablon



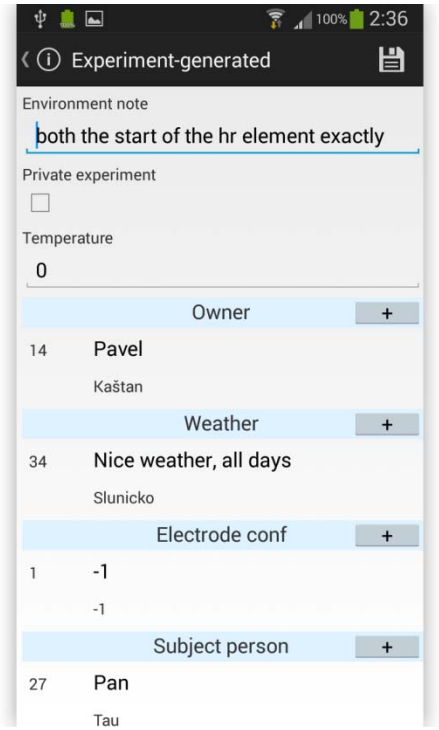
Otevřením šablony se načtou záznamy z databáze. Aby se nemuseli zdlouhavě načítat při každém otevírání šablony, tak jsou uloženy ve statické proměnné workspaceAdapters ve třídě `ListAllFormsFragment`. Výhodou je, že jsou již načteny i pro ostatní šablony daného formuláře nebo pokud šablona obsahuje

podformulář daného typu. Proměnná workspaceAdapters je typu SparseArray, kde klíčem je ID pracovního prostoru a hodnota je HashMap<String, FormAdapter>. Zde je klíčem jméno šablony a hodnota je adaptér obsahující záznamy daného formuláře. V případě přidání dat se aktualizuje pouze tento adaptér.

Obrázek 19 – Seznam záznamů formuláře

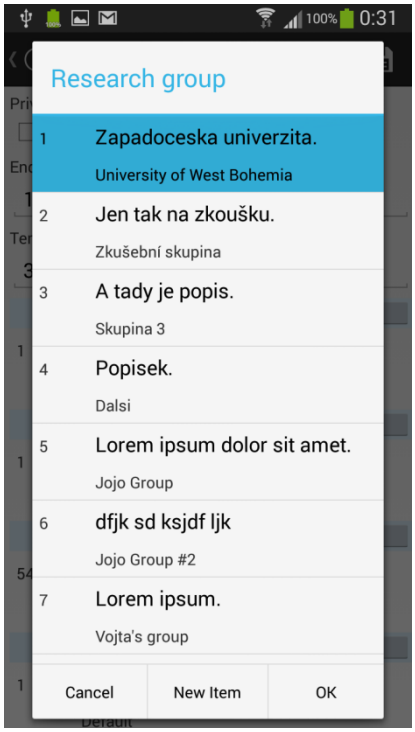


Obrázek 20 – Detail šablony



Dlouhým kliknutím lze vyvolat kontextové menu, kde můžeme zvolit odstranění záznamu nebo změnit hodnoty, které se zobrazují v náhledu. Kliknutím na záznam se zobrazí detail šablony (obrázek 20). Pokud šablona obsahuje podformulář, tak je položka modře zvýrazněná. Záznamy podformuláře jde přidávat tlačítkem +, čímž se vyvolá seznam všech dostupných dat (obrázek 21). Označením položky a potvrzením tlačítkem OK se provede přidání. Kliknutím na tlačítko *New Item* se zobrazí detailní šablona podformuláře. Typ šablony lze změnit v editoru šablon. Vytvořením nového záznamu se zavolá stávající Aktivita metodou startActivityForResult a jako návratová hodnota se předá metodou onActivityResult ID nového záznamu formuláře.

Obrázek 21 – Přidání podformuláře



Editor šablon

Jedna z důležitých funkcí aplikace je možnost editace šablon uživatelského rozhraní. Pro tyto účely jsem vytvořil editor, který pracuje ve dvou režimech. Editor se spustí kliknutím na tlačítko Edit Layout (ikona tužky) na obrázku 19. První režim slouží k editaci vlastností pole, kterou vyvolám dlouhým kliknutím na pole (obrázek 22). Pole mohu odstranit nebo zvolit podrobnější editaci pole, čímž zobrazím Aktivitu, kde mohu měnit vlastnosti pole (například jméno, datový typ).

Obrázek 22 - Editace pole na šabloně

Obrázek 23 – Přesun položek na šabloně

Druhý režim (obrázek 23) umožňuje přesouvat položky na formuláři (ikona Move fields). Pomocí drag and drop mohu pole prohazovat nebo přidat nový sloupec přesunutím pole do oblasti na kraji obrazovky. Drag and Drop je povolen v Android SKD od verze 14. Třída `ViewBuilder` generuje formulář podle definice v odML šabloně. Jednotlivá pole na formuláři jsou instance třídy `UIElement`. V metodě setEditor() se registrují posluchače pro vyjmutí pole v třídě `LayoutTouchListener` zděděná od třídy `OnTouchListener`. V této třídě je nastaven objekt ClipData, do kterého je zkopírováno přesouvané pole. Dále se zde definují cílové oblasti, kam můžeme pole přesouvat. Jakmile se pole přesune na cílovou oblast, tak se zavolá třída `LayoutDragListener`, ve které se zpracuje samotný přesun v metodě onDrag(). V této metodě lze rozlišit jednotlivé typy přenosových událostí. Uvedeny jsou v následující ukázce:

```
public boolean onDrag(View viewB, DragEvent event) {

        int action = event.getAction();
        ViewGroup wrapLayoutB = (ViewGroup) viewB.getParent();

        switch (event.getAction()) {
                case DragEvent.ACTION_DRAG_STARTED:
                        // Co se má dělat na začátku přesunu
                        break;

                case DragEvent.ACTION_DRAG_ENTERED:
                        // Co se má provést při najetí na cílové pole
                        // Nastavuji zvýraznění cílového pole
                        wrapLayoutB.setBackgroundDrawable(enterShape);
                        break;

                case DragEvent.ACTION_DRAG_EXITED:
                        //ukončení přenosu, resetuji zvýraznění
                        wrapLayoutB.setBackgroundDrawable(normalShape);
                        break;
                case DragEvent.ACTION_DROP:

                       /* přesunutí pole na cílové pole, čímž dojde
                          k prohození těchto polí. V případě přesunu do
                          oblasti na okraji obrazovky se přidá pole na
                          řádku a odstraní se z jeho původní pozice */
}
```

Přidání nového pole na šablonu

V editoru je možné přidat nové pole na šablonu kliknutím na ikonku + (Add field). Otevře se Aktivita `FieldEditorAddActivity`, která má dvě záložky (obrázek 24):

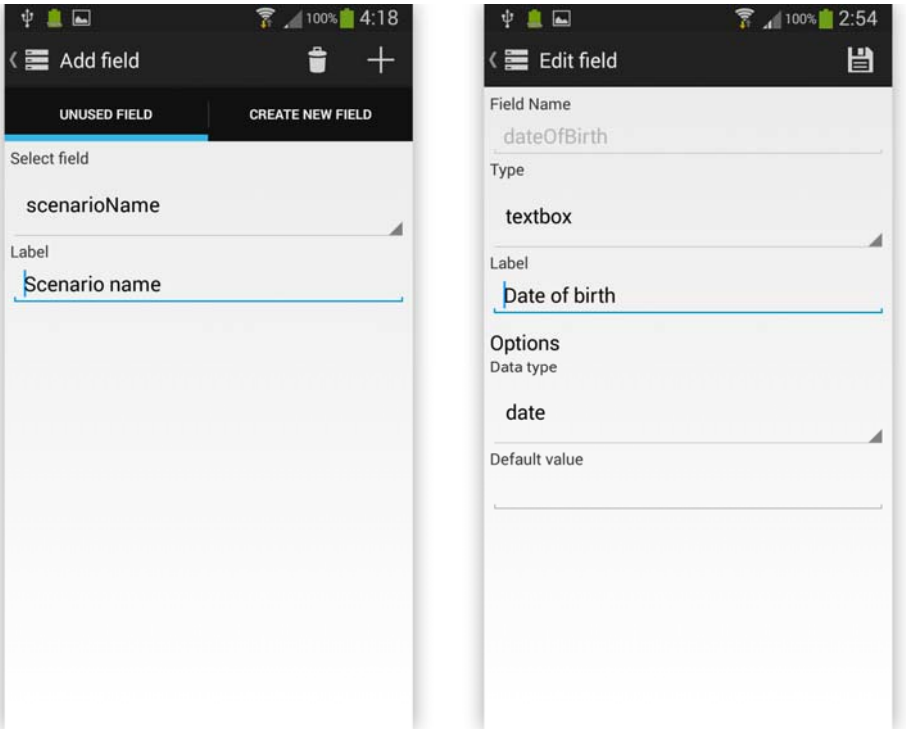
1. Unused Fields – slouží k přidání již existujícího pole formuláře na šablonu
2. Create New Field – vytvoření nového pole formuláře a přidání na šablonu

Při vytváření nového pole formuláře se musí zadat:

1. jméno pole
2. typ pole (textbox, checkbox atd.)
3. label – titulek pole na šabloně
4. volitelné údaje v závislosti na typu pole: datový typ, omezení pole jako je například minimální délka řetězce, maximální zadané číslo atd.,

přidat nebo vytvořit nové pole na formulář.

Obrázek 24 – Přidávání pole na šablonu



6.5 Možná rozšíření

Aplikace podporuje základní objekty uživatelského rozhraní například Textbox, Combobox, Form. Do budoucna je možné rozšířit tuto sadu o komponentu grafu nebo různé ikony.

Na serveru EEGbase není vytvořeno API pro aktualizaci nebo odstranění již existujících záznamů formuláře. Záznamy lze pouze vytvářet, dále při vytváření záznamů není zajištěna kompatibilita. Například záznam nelze uložit, protože neobsahuje požadované pole a informace o tomto poli chybí v serverem generované šabloně. S ukládáním polí souvisí další námět a to je verzování záznamů na serveru aby se aktualizovali pouze novější verze.

6.6 Testování aplikace

Součástí Android SDK jsou nástroje určené pro ladění a testování aplikace. Jedním z nich je nástroj Android Debug Bridge (adb), který nám umožňuje komunikovat s připojeným zařízením nebo emulátorem. Nástroj DDMS slouží jako grafické rozhraní k připojení adb a je integrované do prostředí Eclipse. DDMS nám umožňuje získávat informace o logování, procesech, haldě, souborech a také umožňuje zobrazit databázi. Tohoto faktu bylo využito při ladění databáze, obzvlášť při hledání chyb ve struktuře a v počáteční fázi vývoje, dále při odhalování paměťových úniků dat, v průběhu testování tříd `AsycTask`. Logovací systém Androidu zobrazuje ladící informace systému. Obsahuje systémové zprávy a hlavně výpisy chyb. Uvedené nástroje byly využity při testování aplikace.

Mobilní aplikace byla vyvíjena a testována na zařízení Samsung Galaxy S3, Google Nexus S a pomocí Android emulátoru.

JUnit[45]

Jednotkové testy jsou napsané pomocí knihovny Android JUnit, který rozšiřuje JUnit 3. Aby v testech bylo možné přistupovat ke kontextu aplikace, je třeba dědit od třídy `InstrumentationTestCase`. V balíku data se nacházejí scénáře pro testování databáze, mezi které patří například vytváření, mazaní a aktualizace dat v jednotlivých tabulkách. Pro zajištění nezávislosti jednotlivých testů je vytvořena testovací databáze *eegMobileDatabase.db*, která je umístěná ve složce Assets. Třída `DatabaseCreator` načte databázi *eegMobileDatabase.db* a přepíše databázi aplikace. Pro přístup k ORMlite je vytvořen objekt třídy `DAOFactory` v metodě setUp(), kterému je předán kontext aplikace pomocí metody getTargetContext().

[45] http://developer.android.com/tools/testing/testing_android.html

Robotium[46]

Bylo provedeno automatizované testování uživatelského rozhraní nástrojem Robotium. Vytvořil jsem scénáře pro:

- Vytváření/odstraňování pracovních prostorů
- Přidání nového formuláře a šablony
- Otevření šablony
- Odeslání a stažení dat
- Otevření detailu šablony
- Přesun položky v editoru
- Přidání nového pole na šablonu

Provedené testy jsou dostupné v příloze na CD.

Testování webových služeb[47]

Na testování webových služeb byl použit nástroj SoapUI v 5.0. Testy pokrývají REST API na serveru EEGbase.

Uživatelské testování

Pro potřebu uživatelského testování jsem využil skupinu sedmi osob, která byla seznámena s aplikací. Následovalo pozorování jejich chování při používání aplikace. Byly objeveny tři zásadní nedostatky aplikace:

1. Komplikovaný formulář na přidávání nového formuláře a šablony uživatelského rozhraní, který na jednom místě obsahoval položky pro přidávání nového formuláře, šablony, přidávání polí na šablonu a volbu polí pro náhled záznamů.

[46] Robotium. *Robotium* [online]. [cit. 2014-08-7]. Dostupné z: http://code.google.com/p/robotium/

[47] SoapUI. *SoapUI* [online]. [cit. 2014-08-7]. Dostupné z: http://www.soapui.org/

Mnoho informací na jednom místě zesložiťovalo orientaci a výsledkem bylo přesunout funkcionalitu na místa, kde je uživatelé očekávali.

- Tlačítko pro přidávání nového pole na šablonu se přesunulo do editoru šablony.
- Editace polí sloužící pro náhled se vyvolá jako kontextové menu náhledové položky.

2. Formulář sloužící pro přidávání pole na šablonu se po vytvoření pole uzavřel. Pokud chtěl uživatel vytvářet nové pole, tak ho musel znovu otevřít. Řešením bylo formulář nezavírat, pouze vynulovat vyplněné údaje. Ve chvíli kdy uživatel již nechce přidávat další pole, tak formulář uzavře.
3. Požadavek na snadné přidávání šablon formulářů ve zvoleném pracovním prostoru. Řešením bylo vytvořit formulář obsahující seznam všech dostupných šablon. Uživatel si zvolí, kterou šablonu chce přidat a nebo odstranit.

7 Závěr

Cílem diplomové práce bylo vytvořit mobilní aplikaci pro sběr experimentálních dat a metadat. Vývoj aplikace byl postaven na základě požadavků neurovědeckých organizací spolupracujících s univerzitou. Hlavní požadavky na funkcionalitu aplikace byly následující: Umožnit vytvořit vlastní šablony uživatelského rozhraní formuláře nebo pracovat s již hotovými. Další funkcí aplikace je editace těchto šablon. Po vytvoření šablony uživatelského rozhraní formuláře, může uživatel zaznamenávat data experimentů.

Aplikace řeší problém sdílení dat mezi heterogenními systémy. Po analýze požadavků jsme zvolili Android platformu. Aplikace je navržena pro všeobecné využití, což znamená, že ji mohou ocenit nejen uživatelé z oblasti neurovědy.

Podstatné bylo navrhnout vhodný formát pro společný přenos dat a struktur šablon uživatelského rozhraní. Na základě analýzy jsme zvolili odML formát, který definuje strukturu přenášených dat. Formát přenosu je realizován pomocí XML. Výhodou odML je možnost společného přenosu dat i šablon.

Mobilní aplikace je schopná data synchronizovat se serverem pomocí RESTful webových služeb. Mobilní klient je schopen samostatné práce bez nutnosti připojení k síti. Serverová část byla vyvíjena paralelně s touto diplomovou prací, kde se podařilo vyvinout generátor šablon uživatelského rozhraní, jenž je platformě nezávislý.

Klientskou aplikaci jsem otestoval na několika mobilních zařízeních. Komunikace se serverem byla otestována na systému pro ukládání a řízení dat EEGbase. Testy byly vytvořeny pomocí SOAPui. K otestování databáze a vytváření odML struktry byly vytvořeny Android JUnit testy. Uživatelské rozhraní bylo otestováno pomocí Robotium. Na základě uživatelského testování aplikace byly provedeny změny v ovládání aplikace.

Aplikace byla programována s možností budoucího rozvoje. Je zde například možnost rozšířit ji o další grafické komponenty formuláře a rozšířit funkcionalitu editoru.

8 Použitá literatura

[1] MCWHERTER, Jeff a Scott GOWELL. *PROFESSIONAL MOBILE APPLICATION DEVELOPMENT*. Indianapolis, Indiana: John Wiley & Sons, Inc., 2012. ISBN 978-1-118-20390-3.

[2] DUFFY, Thomas J. Programming with Mobile Applications: Android(TM), iOS, and Windows Phone 7. UK: Cengage Learning, 2013. ISBN 978-1133628132.

[3] VÁVRŮ, Jiří a Miroslav UJBÁNYAI. Programujeme pro Android. Česká Republika: Grada, 2013. ISBN 978-80-247-4863-4.

[4] MCWHERTER, Jeff a Scott GOWELL. Professional Mobile Application Development. England: Wrox, 2013. ISBN 978-1118203903.

[5] Grewe J, Wachtler T and Benda J (2011) A bottom-up approach to data annotation in neurophysiology. *Front. Neuroinform.* **5**:16. doi: 10.3389/fninf.2011.00016

[6] DAHANNE, Anthony. *Instant Spring for Android Starter*. USA: Packt Publishing, 2013. ISBN 1782161902.

[7] SESSA, Carlos. *50 Android Hacks*. USA: Manning Publications, 2013. ISBN 978-1617290565.

[8] PHILLIPS, Bill. Android Programming: The Big Nerd Ranch Guide (Big Nerd Ranch Guides). USA: Big Nerd Ranch Guides, 2013. ISBN 978-0321804334.

[9] Incf. INCF. INCF [online]. 2012 [cit. 2014-06-20]. Dostupné z: http://incf.org/

[10] NIF. NIF. NIF [online]. 2012 [cit. 2014-06-20]. Dostupné z: https://www.neuinfo.org/

[11] Irods. IRODS. Irods [online]. 2010 [cit. 2014-06-20]. Dostupné z: https://irods.org/

[12] G-node. G-NODE. G-node [online]. 2014 [cit. 2014-06-20]. Dostupné z: http://www.g-node.org/about

[13] Open metadata Markup Language. G-NODE. G-node [online]. 2014 [cit. 2014-08-15]. Dostupné z:http://www.g-node.org/projects/odml/

[14] APACHE CORDOVA. *Apache Cordova* [online]. 2012 [cit. 2014-08-15]. Dostupné z: http://cordova.apache.org/

[15] PLUGINS REGISTRY. APACHE CORDOVA. *Apache Cordova* [online]. 2012 [cit. 2014-08-15]. Dostupné z:http://plugins.cordova.io/#/

[16] Developer.android. *Developer.android* [online]. 2010 [cit. 2014-06-20]. Dostupné z:http://developer.android.com/about/dashboards/index.html

[17] Spring for Android Reference Documentation. *Spring.io* [online]. 2014 [cit. 2014-06-20]. Dostupné z:http://docs.spring.io/spring-android/docs/1.0.2.BUILD-SNAPSHOT/reference/htmlsingle/

[18] Www.vogella.com: Multi-pane development in Android with Fragments - Tutorial. *Vogella* [online]. 2013 [cit. 2014-06-21]. Dostupné z: http://www.vogella.com/tutorials/AndroidFragments/article.html

[19] *Statcounter* [online]. 1999 [cit. 2014-06-20]. Dostupné z: http://gs.statcounter.com/

[20] PETR, Ježek a Mouček ROMAN. *Framework for automatic generation of graphical layout compatible with multiple platforms* [online]. 15-19 Sept. 2013 [cit. 2014-06-20].

[21] MIKO, Petr. *Mobile system for management of EEG/ERP experiments*. Plzeň, 2013. Master Thesis. University of West Bohemia.

[22] DUDOVÁ, Veronika. *Rozvrh hodin pro mobilní zařízení*. Plzeň, 2012. Diplomová práce. Západočeská univerzita v Plzni. Vedoucí práce Ing. Pavel Herout, Ph.D.

9 Seznam zkratek

Zkratka	Význam
NIF	The Neuroscience Information Framework
INCF	International Neuroinformatics Coordinating Facility
G-Node	German Neuroinformatics Node
EEG	ElektroEncephaloGraphy
ERP	Even-Related Potentials
OHA	Open Handset Aliance
ASL	Apache Software License
GPL	General Public Licence
SDK	Software Developer Kit
DVM	Dalvik Virtual Machine
API	Application Programming Interface
XML	Extensible Markup Language
HTML	HyperText Markup Language
XAML	Extensible Application Markup Language
JSON	Java Script Object Notation
YAML	Yaml Ain't Markup Language
GUI	Graphical Youser Interface
odML	Open MetaData Markup Language
SOAP	Simple Object Access Protocol
REST	Representation State Transfer
JAXB	Java Architecture for XML Binding
POJO	Plain Old Java Object
CRUD	Create Read Delete Update

10 Přílohy

Příloha 1

	amazon-fireos	android	blackberry 10	Firefox OS	ios	Ubuntu	wp7 (Windows phone 7)	wp8 (Windows phone 8)	win 8 (Windows 8)	tizen
cordova CLI	✓Mac, Windows, Linux	✓Mac, Windows, Linux	✓Mac, Windows	✓Mac, Windows, Linux	✓Mac	✓Ubuntu	✓Windows	✓Windows	✓	✘
Embedded WebView	✓(see details)	✓(see details)	✘	✘	✓	✓	✘	✘	✘	✘
Plug-in Interface	✓(see details)	✓(see details)	✓(see details)	✘	✓(see details	✓	✓(see details)	✓(see details)	✓	✘
Platform APIs										
Accelerometer*	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Camera*	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Capture*	✓	✓	✓	✘	✓	✓	✓	✓	✘	✘
Compass*	✓	✓	✓	✘	✓(3GS+)	✓	✓	✓	✓	✓
Connection*	✓	✓	✓	✘	✓	✓	✓	✓	✓	✓
Contacts*	✓	✓	✓	✓	✓	✓	✓	✓	✘	✘
Device*	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Events*	✓	✓	✓	✘	✓	✓	✓	✓	✓	✓
File*	✓	✓	✓	✘	✓	✓	✓	✓	✓	✘
Geolocation*	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Globalization*	✓	✓	✘	✘	✓	✓	✓	✓	✘	✘
InAppBrowser*	✓	✓	✓	✘	✓	✓	✓	✓	uses iframe	✘
Media*	✓	✓	✓	✘	✓	✓	✓	✓	✓	✓
Notification*	✓	✓	✓	✘	✓	✓	✓	✓	✓	✓
Splashscreen*	✓	✓	✓	✘	✓	✓	✓	✓	✓	✘
Storage*	✓	✓	✓	✘	✓	✓	✓LocalStorage & IndexedDB	✓LocalStorage & IndexedDB	✓LocalStorage & IndexedDB	✓