VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

APLIKACE PRO VIZUALIZACI SENZORICKÝCH DAT V OPERAČNÍM SYSTÉMU ANDROID

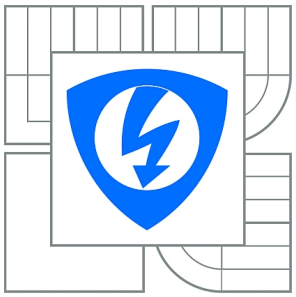
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE PETR SEDLÁČEK
AUTHOR

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

APLIKACE PRO VIZUALIZACI SENZORICKÝCH DAT V OPERAČNÍM SYSTÉMU ANDROID

APPLICATION OF SENSOR DATA VISUALIZATION IN ANDROID OS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

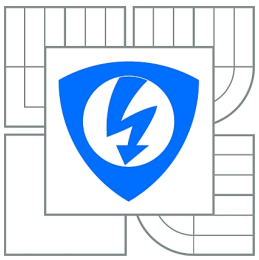
AUTOR PRÁCE — PETR SEDLÁČEK

AUTHOR

VEDOUCÍ PRÁCE — Ing. MILAN ŠIMEK, Ph.D.

SUPERVISOR

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Petr Sedláček	***ID:***	148149
Ročník:	3	***Akademický rok:***	2013/2014

NÁZEV TÉMATU:

Aplikace pro vizualizaci senzorických dat v operačním systému Android

POKYNY PRO VYPRACOVÁNÍ:

Cílem bakalářské práce je vytvoření aplikace v pro operační systémy Android. Hlavní funkcí aplikace je načtení dat z webového cloudu Xively a zobrazení uzlů bezdrátové senzorové sítě v přehledném formátu na chytrém telefonu. Aplikace musí být schopná zobrazit senzory a uvést aktuální a historické naměřené hodnoty. Pro zobrazení těchto dat je uživatel vyzván k zadání přístupového klíče. Dále bude aplikace disponovat funkcí filtrování dat, vyhledávání senzorových uzlů, zobrazování grafů a jejich export.

DOPORUČENÁ LITERATURA:

[1] Stojmenovic I., Handbook of Sensor Networks, Wiley,ISBN:13 978-0-471-68472-5, 2005.
[2] FARAHANI, Shahin. Zigbee Wireless Networks and Transceivers. [s.l.] : Elsevier, 2008. 329 s. ISBN 978-0-7506-8393-7

Termín zadání:	10.2.2014	***Termín odevzdání:***	4.6.2014

Vedoucí práce: Ing. Milan Šimek, Ph.D.
Konzultanti bakalářské práce:

doc. Ing. Jiří Mišurec, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Cílem bakalářské práce je vytvoření aplikace pro operační systém Android, jenž má za úkol stáhnout z webového cloudu Xively uložená senzorická data. Práce obsahuje praktickou a teoretickou část. První kapitola se zaměřuje převážně na teorii týkající se dané problematiky a obsahuje pojmy nezbytné pro další popis vývoje. Obsahuje také stručný popis systému Android z hlediska funkčnosti a jeho historie. Uvedené pojmy jsou zde však uvedené pouze ve stručnosti a slouží pouze k seznámení s touto problematikou.
Druhá kapitola je zaměřena na popis vývoje aplikace, jenž tvořila výstup semestrálního projektu. Třetí kapitola obsahuje popis vývoje výsledné verze aplikace, jenž svoji funkčností vychází z aplikace vytvořené v rámci semestrálního projektu. V obou kapitolách je poměrně detailně uveden popis postupného vývoje aplikace včetně ukázek důležitých či zajímavých částí použitého kódu. Ve třetí kapitole jsou také uvedeny obrázky výsledné aplikace včetně jejich popisu. V závěru jsou poté shrnuty dosažené výsledky bakalářské práce a jejich srovnání se zadáním práce.

KLÍČOVÁ SLOVA

Android, aplikace, softwarový vývoj, senzory, senzorická data, vizualizace dat, HTTP komunikace, webový cloud

ABSTRACT

The aim of this Bachelor thesis is to create an application for the Android operating system which purpose is to download saved sensor data on a Xively web cloud. The thesis has a practical and theoretical part. The first chapter is mainly focused on theory that is necessary for further description of the development process. It also contains brief introduction to funcionality and history of the Android operating system. The theory introduced here is very brief and its purpose is solely to introduce the reader to the given subject.
The second chapter focuses on describing development of basic version of the application which represented the outcome of the semestral project. The third chapter describes development of the final version of the application which enhances the basic application. In both chapters the development is described in a chronological order with examples of important or interesting parts of the used code. The third chapter also contains several pictures of the final application inluding their description. The conclusion sums up all achieved results of this thesis and their comparison to the given assignments.

KEYWORDS

Android, application, software development, sensors, sensor data, data visualization, communication using HTTP, web cloud

SEDLÁČEK, Petr *Aplikace pro vizualizaci senzorických dat v operačním systému Android*: bakalářská práce. BRNO: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2014. 43 s. Vedoucí práce byl Ing. Milan Šimek, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Aplikace pro vizualizaci senzorických dat v operačním systému Android“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

BRNO ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Milanu Šimkovi, Ph.D. za cenné rady, zkušenosti a možnost podílet se tvorbou této bakalářské práce na mezinárodním projektu s názvem „Telecalm“.

BRNO ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

Tato bakalářská práce se zaměřuje, jak již z názvu vyplývá, na vývoj aplikace pro operační systém Android. Aplikace, na kterou je tento projekt zaměřen, si klade za cíl dát uživateli možnost si v nějaké formě na svém mobilním zařízení zobrazit senzorická data, která jsou uložená na webovém cloudu. S funkčním internetovým připojením má tedy uživatel možnost si data zobrazit v podstatě kdekoliv a kdykoliv. Účel aplikace je tedy hlavně zobrazení dat, nikoliv jejich upravování či zapisování. V práci je také kladen důraz na dostatečnou univerzálnost aplikace, tedy její použití pro různé typy dat a pro jakéhokoliv uživatele. Podmínkou pro použití aplikace je tedy pouze vlastnictví účtu na webovém cloudu.

V první kapitole jsou popsány teoretické základy nutné pro následující praktický popis práce. Je zde rozebrán webový cloud, komunikace s ním a také důvod jeho použití. Dále taktéž základní pojmy ve vývoji pro platformu Android a také zjednodušený pohled na jeho architekturu. Je zde také stručně uvedena historie tohoto operačního systému.

Druhá kapitola se již orientuje na samotné praktické provedení práce. Kapitola zahrnuje popis vývoje základní verze aplikace, která tvořila výstup semestrálního projektu. V další kapitole následuje popis vývoje finální verze aplikace, jenž vychází z aplikace základní. V obou kapitolách je uveden detailní popis vývoje a to jak z hlediska programového, včetně ukázek počítačového kódu, tak z hlediska funkčnosti. Třetí kapitola navíc obsahuje i obrázky zachycující výslednou podobu aplikace. Obě tyto kapitoly jsou psány chronologicky, tedy popisují vývoj v postupném časovém sledu. Samotný závěr poté shrnuje dosažené výsledky a jejich srovnání se zadáním práce.

1 TEORETICKÝ ÚVOD

Tato kapitola je zaměřena převážně na teoretický popis práce, tedy základní pojmy, které je potřeba znát pro pochopení dané problematiky a také důvody použití zvolených prostředků k realizaci práce.

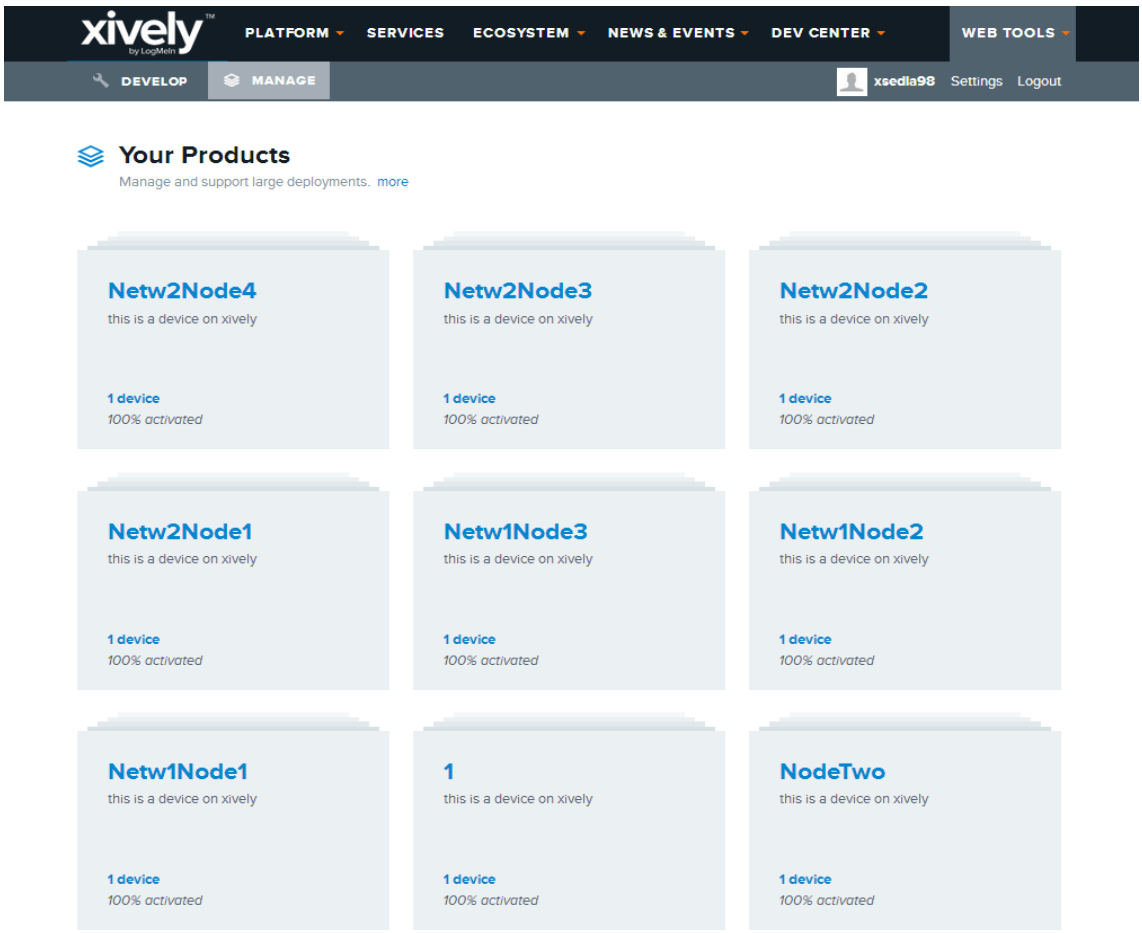
1.1 Webový cloud Xively

Na úvod je nutno říci, jak funguje a co je vlastně systém Xively. Jde o webový cloud, ve kterém má uživatel uložena nějaká data, vetšinou z měření nějakých veličin, například teploty či vlhkosti. Cloud neboli Cloud computing je koncept, který umožňuje uživateli přístup k jeho datům uloženým na některém ze serverů daného cloudu. K těmto datům má nepřetržitý přístup, pokud má samozřejmě aktivní internetové připojení. Připojení do cloudu může probíhat například pomocí internetového prohlížeče. Samotná komunikace mezi servery Xively a uživatelem probíhá pomocí HTTP příkazů, takže pokud chce uživatel data ze serveru číst, použije příkaz HTTP GET a pokud chce nová data ukládat, použije příkaz HTTP POST. Výhody použití cloudu Xively jsou tedy zřejmé. Uživatel ke svým datům může mít kdykoliv přístup a objem dat zatěžuje pouze servery daného cloudu, nikoliv však uživatele. Služba může být buď zdarma, která je však omezená nebo placená, která nabízí mnohem více možností. Xively používá strukturu tzv. „Internet of Things“, což je soubor nějakých fyzických objektů, které mají svoji virtuální reprezentaci v síti, která se svojí stavbou podobá Internetu. Samozřejmě daný objekt může být i čistě virtuální. Při práci na této bakalářské práci byly používány především virtuální objekty.

Když už je jasné, jak vypadá struktura Xively a jakým způsobem probíhá komunikace, musí být ještě popsáno, s jakými zařízeními Xively pracuje. Úkolem této práce bylo pracovat s virtuálními senzory, které snímají teplotu a vlhkost na nějakém místě. Tyto senzory byly rozmístěny na mapě v aplikaci „Outdoorview“, kterou vytvořil tým Wislab, sídlící na Ústavu telekomunikací na VUT v Brně. Odkaz na tuto apliaci je zde: http://wislab.cz/wsnapp/outdoorview/index.htm.

Na webovém cloudu je dále nutno získaná data nějakým způsobem od sebe odlišit. Musí být například rozlišeno jaká data se týkají teploty, jaká vlhkosti či jiné měřené veličiny a také se musí odlišit samotné senzory, které data získávají a ukládají na cloud. Každý uzel v síti (senzor) je reprezentován pomocí tzv. „feedu“. Každý fyzický senzor má tedy svoji virtuální reprezentaci v podobě feedu. Stejným způsobem jsou od sebe odlišeny jednotlivé veličiny, jejichž virtuální reprezentace se nazývá „datastream“. Každý uzel je navíc definován unikátním „API klíčem“, dále svým

„Feed ID“ a taky URL adresou v této podobě: „https://api.xively.com/v2/feeds/ID[1]“, kde ID je unikátní ID daného uzlu. Dále mohou být u uzlu definována tzv. „metadata“, což jsou data navíc, která informují například o datu a času vytvoření daného uzlu, dále o uživateli, pod kterého daný uzel spadá a mohou také obsahovat další unikátní data. Příklad toho může být například identifikátor „outdoorview“ v položce „tags“, která je součástí metadata. Tento identifikátor nás informuje o tom, že daný uzel byl vytvořen ve výše zmíněné aplikaci Outdoorview. Uzly vytvořené v rámci bakalářské práce jsou uvedeny na obrázku č.1.1.



Obr. 1.1: Virtuální uzly vytvořené pomocí aplikace Outdoorview

Dále je také potřeba vysvětlit princip ochrany a zabezpečení v systému Xively. Uzly se v aplikaci Outdoorview vytváří pomocí tzv. „API Master Key“, což je identifikátor, pod nějž spadají všechny v něm vytvořené uzly. Identifikátor je vždy samozřejmě zcela unikátní. Tomuto klíči je možno také přiřadit různá práva, jako je buď pouze čtení, zápis, vytváření a mazání nebo vše dohromady. Dále je u něj také možno povolit tzv. „Private access“, který umožní přistupovat do privátních uzlů.

[1]Bývá také označován jako API Endpoint.

Uzly je totiž také možně rozdělit na veřejné a privátní. Pokud bychom znali URL adresu daného uzlu, avšak daný uzel by byl privátní, systém Xively by odmítnul přístup bez správné autorizace uživatele. Vytvořené veřejné uzly lze sice zobrazit, avšak pouze v případě, že má uživatel účet na Xively a je přihlášen. Xively dále podporuje při komunikaci používání zabezpečeného protokolu HTTPS, aby během přenosu nebylo možné danou komunikaci neoprávněně odposlouchávat.

Cílem tohoto úvodu bylo hlavně stručně popsat jak funguje systém Xively a jeho jednotlivé části, které jsou potřeba pro vypracování bakalářské práce. Mnohem více bližších informací o tom, jak Xively funguje a různé další aspekty komunikace lze najít na stránkách Xively: [8].

1.2 Systém Android

V dnešní době zažívá vývoj aplikací obrovský rozmach. Systém Android má údajně v současné době již přes miliardu uživatelů a tento počet neustále roste. Android také na svých stránkách uvádí, že denně je na světě aktivováno až milion zařízení běžících na tomto operačním systému. Z tohoto čísla je tedy patrné, proč je o vývoj aplikací na tuto platformu takový zájem. S různými formami aplikací se setkává každý uživatel chytrého telefonu, ať už jde o jakýkoliv operační systém.

Nebylo by vhodné zde rozebírat výhody či nevýhody jednotlivých systémů neboť tato práce je zaměřena pouze na operační systém Android. Android je už od doby svého vzniku nesmírně populární a to hlavně díky jednoduchosti a hlavně faktu, že jde o open-source platformu. Společnost Google koupila firmu Android Inc. v roce 2005 a v roce 2007 byla založena tzv. „Open Handset Alliance“, což je společnost sdružující velké množství firem, které tehdy oznámily cíl vytvořit nový otevřený standard pro mobilní telefony. První telefon běžící na operačním systému Android byl telefon „HTC Dream“.

V dnešní době je operační systém Android využíván největšími výrobci mobilních telefonů a je nesmírně populární zejména pro vývoj aplikací. Začít vyvíjet aplikace pro tuto platformu je s alespoň mírnými zkušenostmi v programovacím jazyce Java poměrně snadné. Všechny softwarové nástroje, tutoriály a ukázky kódu jsou volně k dostání na stránkách [1].

Jako platforma pro vytvoření aplikace byl zvolen systém Android, neboť autor práce je vlastníkem mobilního telefonu HTC One S s operačním systémem Android ve verzi 4.1.2. Dále bylo nutno použít správné vývojové prostředí pro tvorbu aplikace. V tomto ohledu je pravděpodobně nejlepší volbou vývojové prostředí Eclipse, které využívá objektově orientovaného programovacího jazyka Java a pro které má Android širokou podporu. Existují i jiná alternativní prostředí buď s nebo bez pod-

pory firmy Google. Pro vývoj byl tedy zvolen ADT (Android Development Tools) plugin pro prostředí Eclipse. Tento plugin je volně ke stažení na stránkách [1]. Na těchto internetových stránkách od firmy Google je uvedeno takřka vše, co začínající vývojář pro paltformu Android potřebuje. Od volně stažitelného vývojového prostředí přes úplnou dokumentaci všech příkazů pro vývojové prostředí, výuku základních pojmů a principů, tutoriály pro grafické prostředí až po ukázky konkrétních kódů napsaných v prostředí Eclipse. Stále jsou však na stránkách uvedeny pouze základy programování jednotlivých částí, pro hlubší pochopení a prohloubení znalostí z oblasti vývoje aplikací pro Android je dobré sáhnout po odborné literatuře.

1.3 Základní pojmy ve vývoji pro Android

Samotným začátkem práce tedy bylo stažení balíčku ADT ze stránek [1]. Po instalaci celého balíku ADT je dále nutné ve vývojovém prostředí definovat, na jakém zařízení bude aplikace testována. Vývojové prostředí nabízí buď možnost spuštění aplikace na fyzickém zařízení nebo zde uživatel může definovat virtuální zařízení. Pro spuštění aplikace bylo vytvořeno virtuální zařízení co nejbližší cílovému mobilnímu telefonu, tedy velikost obrazovky $4,7''$ a verzi systému Android 4.1.2 (verze „Jelly Bean[2]“). Prostředí Eclipse po spuštění aplikace spustí tzv. „emulátor“, který simuluje reálný mobilní telefon běžící s danou verzí systému Android. Zde uživatel může danou aplikaci testovat i bez vlastnictví reálného zařízení. Aplikaci je však vždy dobré testovat na reálných zařízeních neboť emulátor není dostatečně schopen simulovat vše. Více o tomto tématu se lze dočíst v [4]. Tento emulátor, neboli „Dalvik Virtual Machine“, byl opět vytvořen společností Google. Vytvořený kód se kompiluje do instrukcí, které jsou hardwarově nezávislé a emulátor je vykoná na daném zařízení. Dalvik je v podstatě obdoba klasické Java Virtual Machine, akorát je optimalizován pro minimální požadavky na paměť [3].

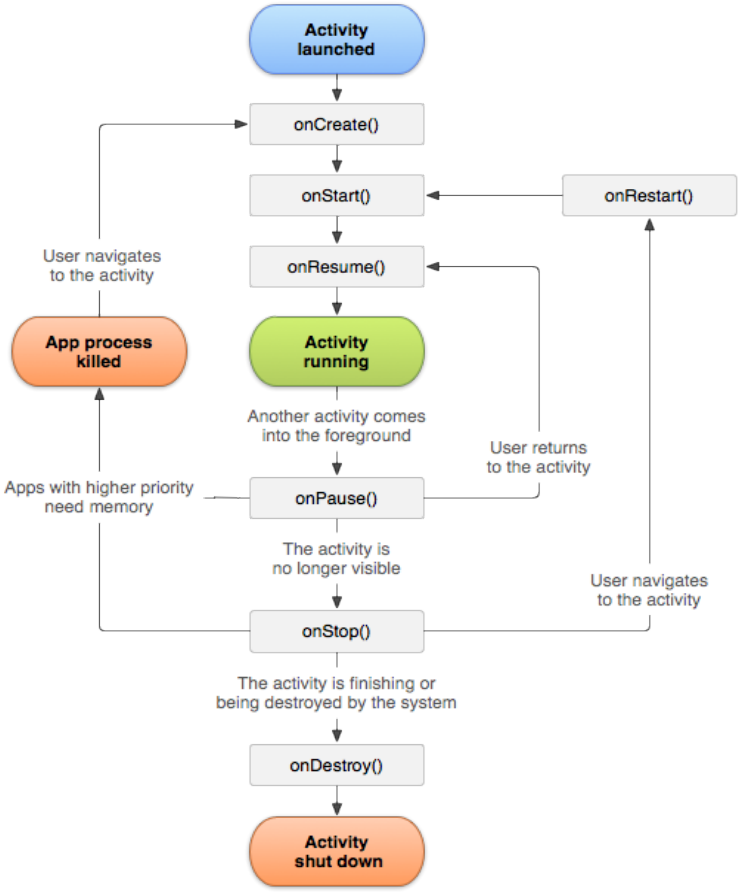
Programování v Android SDK (Software Development Kit) je převážně stejné, jako klasické programování v jazyce Java. Zde se pouze u různých tříd musí dané proměnné provázat s grafickým prostředím, které se vytváří pomocí jazyka XML. Struktura jinak vypadá stejně, jako v jazyce Java. V jednom balíčku je několik tříd, které jsou provázány se soubory ve formátu XML a tzv. „Activities“ neboli Aktivity. Aktivita reprezentuje momentální činnost aplikace a její interakci s uživatelem. V aplikaci může během jednoho okamžiku běžet pouze jedna Aktivita a je to právě ta, kterou momentálně uživatel vidí a používá ji. Uživatel mezi jednotlivými Aktivitami přepíná. Každá třída správně reprezentuje jednu Aktivitu aplikace. Aktivita ve

[2]Verze se také označují pomocí API Levelu a v současnosti je jich celkem 19. První verze Jelly Bean odpovídá API Levelu 16.

výchozím stavu implementuje několik předem definovaných metod. Konkrétně jde o metody **onCreate()**, **onStart()**, **onPause()**, **onResume()**, **onStop()** a **onDestroy()**. Jak z názvu metod vyplývá, každá z nich určuje, co má v kterém stavu aplikace dělat. Celkem detailně je tento „životní cyklus“ Aktivit zobrazen na obrázku č.1.2. Metoda **onCreate()** se zavolá, když je aplikace poprvé spuštěna. Poté aplikace přejde do metody **onStart()** a poté do **onResume()**. Pokud se do popředí dostane nějaká jiná aplikace či pokud uživatel z aplikace vyjede, ale nezastaví ji úplně, zavolá se metoda **onPause()**. Pokud je dostatek místa ve vyrovnávací paměti a uživatel se do aplikace vrátí, aplkace přejde zpět do metody **onResume()**. Pokud k vrácení do aplikace nedojde či pokud není dostatek místa v paměti, zavolá se metoda **onStop()** a poté metoda **onDestroy()**, čímž se aplikace zcela zastaví a odstraní z paměti. Je důležité právě v momentech, kdy se jiná aplikace dostane do popředí či když je sledovaná aplikace ukončována, aby se uložila důležitá data, která mohou být jinak nenávratně ztracena. Více o tomto tématu se lze dočíst v [1].

Systém Android je postavený na Linuxovém jádře, což mu právě umožňuje hardwarovou nezávislost na zařízení. Android používá Linux pro správu paměti, správu procesů, síťovou komunikaci atd. Pokud bychom si představili strukturu systému Android ve vrstvách, nejspodnější vrstvu zastává právě Linux, který pracuje se samotným hardwarem. Hned nad touto vrstvou se nachází tzv. „Nativní knihovny“, které jsou psány v jazyce C a C++ a kompilují se vždy pro právě používaný hardware. V této vrstvě se také nachází **Android Runtime**, který má na starosti právě správu emulátoru. Nad těmito vrstvami se nachází **Application Framework**. Tato vrstva spravuje životní cyklus aktivit a také obsahuje samotné nástroje pro pro vytváření aplikací. Obsahuje grafické objekty i systémové nástroje. Nejvyšší je vrstva **Apllications and Widgets**. S touto vrstvou přímo pracuje sám uživatel a je jako jediná běžnému uživateli přístupná. Jak z názvu vypovídá, tato vrstva obsahuje samotné nainstalované aplikace a widgety, což jsou aplikace, které běží pouze na jedné z domovských obrazovek. Při koupi telefonu je v systému nainstalováno několik defaultních aplikací nutných pro základní běh telefonu (email, webový prohlížeč atd.), které může uživatel dále rozšiřovat stahováním nových aplikací z virtuálního obchodu Google Play, kde mohou i amatérští vývojáři zdarma nabízet svoje aplikace ke stažení [3].

Dalším důležitým nástrojem pro tvorbu aplikací jsou tzv. „Intenty“. Jsou to nástroje, které umožňují komunikaci nejen mezi Aktivitami. Pomocí Intentů je možno mezi jednotlivými Aktivitami přepínat a také se dá pomocí nich mezi Aktivitami přeposílat i jednoduchá data typu integer, boolean, String atd. Vždy, když se v aplikaci přepínají Aktivity či když si Aktivity předávají jednoduchá data, používají se výhradně Intenty. Více o Intentech, jejich dělení atd. se lze opět dočíst v [1].



Obr. 1.2: Životní cyklus aktivit (převzato z [1])

2 VÝVOJ ZÁKLADNÍ VERZE APLIKACE

V kapitole č.1 a v úvodu byly popsány cíle a zaměření práce. Tato kapitola je zaměřena na praktické vypracování úvodní části práce, tedy semestrálního projektu.

2.1 Zahájení vývoje

Prvním úkolem bylo vytvoření aplikace, která se připojí ke Xively a stáhne si data z jednoho uzlu a v nějaké formě je zobrazí. Jako nejlepší řešení se jevilo zobrazení dat v tabulce. Pro toto zobrazení je zcela jistě nejlepší použít tzv. „TableLayout“, což je zobrazení, které řadí objekty tzv. „TextView“ do řádků a sloupců. Byla tedy vytvořena malá tabulka, ve které se ve dvou TextView zobrazují hodnoty změřené daným senzorem. Aby bylo možno hodnoty také po čase obnovit, bylo přidáno tlačítko, po jehož stisknutí se opět naváže spojení a hodnoty se znovu zobrazí. Tyto grafické objekty jsou uloženy v souboru **activity_main.xml** a třída, která implementuje toto grafické rozhraní a zároveň zobrazuje příslušnou aktivitu, nese název **Request**. Dále bylo také nutné zajistit autorizaci uživatele a to co v nejlepším možném bezpečném módu. Komunikace je tedy nastavena pomocí příkazů HTTPS a byla vytvořena aktivita **MainActivity**, která je zároveň „hlavní“ aktivitou aplikace. Jsou zde uvedeny uvozovky neboť hlavní je zde pouze obrat. Aplikace musí mít definovanou některou z aktivit jako hlavní, aby mohla fungovat a jako hlavní byla zvolena právě tato aktivita. Zde jsou dvě textová pole, jedno z nich má šifrovaný formát pro zadávání hesla, tlačítko pro potvrzení a navázání spojení a tzv. „CheckBox“, který umožňuje aplikaci ukládat jméno a heslo do souboru. Jak bylo později zjištěno, tento krok byl velmi důležitý neboť Xively vyžaduje, aby při každé HTTP žádosti bylo znovu zasláno jméno a heslo. Tedy zadání těchto parametrů hned na začátku by bylo pro uživatele zbytečné, když by je musel zadávat při každém požadavku.

Ukládání do souboru probíhá pomocí rozhraní (Interface) s názvem SharedPreferences[1]. Rozhraní je implementováno tak, že když uživatel zaškrtne políčko CheckBoxu, zadané parametry se uloží do souboru ve formátu XML. Jak je toho konkrétně dosaženo, je naznačeno ve výpisu kódu č.2.1. Ukládání souboru a čtení ze souboru je navíc nastaveno tak, aby k danému souboru měla přístup pouze tato aplikace, což zvyšuje bezpečnost. K souboru nemá přístup ani sám uživatel, jedině kdyby měl nastavená oprávění typu „root“. Z tohoto souboru si poté mohou ostatní aktivity přihlašovací údaje přečíst a uživatel je tedy nemusí zadávat opakovaně. Pokud uživatel odškrtne políčko CheckBoxu, dojde k promazání údaje budou smazány.

[1] Více o tomto rozhraní lze opět najít v [1].

Aktivita je navíc nastavena tak, že pokud je jméno a heslo v pořádku, aplikace přejde do další Aktivity. Pokud je jméno a heslo zadáno špatně, aplikace uživatele dále nepustí a zobrazí se vyskakovací zpráva, tzv. „Toast“, a upozorní uživatele, že zadal špatně jméno a heslo. Zároveň aplikace kontroluje, zda je CheckBox zaškrtnutý, neboť uložení přihlašovacích údajů je pro funkci aplikace kritické. Pokud tedy CheckBox zaškrtnutý není, opět vyskočí zpráva v podobě „Toastu“ a upozorní uživatele, aby údaje uložil. Vypracování první aplikace tedy bylo poměrně jednoduché, i když časově náročné neboť zprvu se vyskytly problémy s autorizací na straně Xively. Původní řešení obsahovalo pouze tzv. základní autorizaci, která funguje v internetovém prohlížeči.

```
1 if (checkBox.isChecked()) {
2              editor.putBoolean("saveLogin", true);
3              editor.putString("username", username);
4              editor.putString("password", password);
5              editor.commit();
6             } else {
7              editor.clear();
8              editor.commit();
9             }
```

Výpis kódu 2.1: Proces zápisu jména a hesla do souboru

Kompletní komunikace s cloudem Xively je prováděna ve třídě „HttpComm“. Plný výpis jejího kódu je uveden v příloze č.A.1. HTTP komunikaci v systému Android není možné provádět v hlavním vlákně aplikace od verze 3.0 systému Android [1]. Pro tyto účely a pro účely práce aplikace na pozadí byla vytvořena třída typu **AsyncTask**. Dá se říci, že na této třídě celá aplikace stojí. Díky třídě AsyncTask je veškerá komunikace prováděna na pozadí a tedy nijak nezasahuje do běhu hlavního vlákna programu. Třída AsyncTask implementuje několik metod. V aplikaci je využíváno hlavně metody **doInBackground()** a **onPostExecute()**. Aktivita, která chce provést komunikaci se serverem zavolá tuto třídu a předá jí potřebné parametry. Úplný výpis kódu třídy HttpComm je uveden v příloze A.1. Třída HttpComm má definovány tři parametry. První z nich je typu String. Ten udává, v jaké podobě se do metody doInBackground() zadávájí parametry k provedení operace. Další parametr nás může informovat o průběhu vykonání komunikace. Tento parametr zatím ve třídě HttpComm není definovaný neboť pro běh aplikace není nijak stěžejní, je pouze informativní. Poslední parametr určuje návratový typ, který daná třída vrací. Zde je jako návratový typ definován tzv. „JSONObject“. Xively má totiž několik forem pro zobrazení dat. Zobrazení na dané URL adrese tedy může být buď ve formátu CSV, klasickém formátu XML nebo právě ve formátu JSON. Jako návratový typ byl zvolen právě poslední jmenovaný, neboť je pro uživatele celkem přehledný a

je snadné z něj sledované hodnoty získat. Typický výpis JSONObjectu je uveden na obázku č.2.1. Jde o výpis dat z jednoho uzlu uloženého v Xively.


```
{
        "id": 121601,
        "title": "Demo",
        "private": "false",
        "feed": "https://api.xively.com/v2/feeds/121601.json",
        "status": "frozen",
        "updated": "2013-04-23T03:25:48.686462Z",
        "created": "2013-03-29T15:50:43.398788Z",
        "creator": "https://xively.com/users/calumbarnes",
        "version": "1.0.0",
        "datastreams": [
        {
                "id": "example",
                "current_value": "333",
                "at": "2013-04-23T01:10:02.986063Z",
                "max_value": "333.0",
                "min_value": "333.0"},
        {
                "id": "key",
                "current_value": "value",
                "at": "2013-04-23T00:40:34.032979Z"},
        {
                "id": "temp"
        }],
        "location": {
                "domain": "physical"
        }
}
```


Obr. 2.1: Výpis dat z jednoho uzlu ve formátu JSON (převzato z [8])

Dále program pokračuje do metody doInBackground(). Na začátku této metody jsou definovány parametry, které třídě HttpComm předávají ostatní Aktivity. Tyto parametry jsou uživatelské jméno, heslo a URL adresa, kterou chce daná Aktivita zobrazit. Dále probíhá příkaz HTTP GET a autorizace uživatele. Pokud se aplikaci dostane odpovědi od serveru, je do proměnné **result** uložen výsledek operace, tedy v tomto případě JSONObject. Metoda onPostExecute() je zde dá se říci již navíc, do ní pouze metoda doInBackgroud() předává výsledek operace.

Třídy, které pro svoji činnost třídu HttpComm potřebují, nesou název **Request** a **MainActivity**. Činnost třídy MainActivity byla již popsána výše, jakmile proběhne správná autorizace uživatele, aplikace pomocí Intentu přejde do další Aktivity, kterou reprezentuje třída Request. V této třídě je využito výše popsané tabulkové zobrazení pro samotnou vizualizaci dat. Pro obnovení zobrazení dat stačí vždy pouze stisknout tlačítko, kdy se znovu provede příkaz HTTP GET. Pokud vše proběhne

v pořádku, vyskočí při zobrazení hodnot navíc ještě infotmativní zpráva „Success!“ opět v podobě „Toastu“. Velmi důležitým krokem, který de facto používá každá Aktivita v aplikaci, je tzv. „parsing“. V podstatě jde o průchod daného JSON objektu cyklem či více cykly a najití požadované hodnoty a její výpis. Jak dané hodnoty získat je naznačeno ve výpisu kódu č.2.2. Získané hodnoty se poté zobrazí v daných TextView. Třída MainActivity je svázána s grafickou reprezentací v souboru **activity_main.xml** a třída Request se souborem **user_auth.xml**.

```
1  JSONObject result = new HttpComm().execute(params).get();
2
3                  JSONArray array = result.getJSONArray("datastreams");
4                  for (int i=0; i <= array.length(); i++)
5                  {
6                    JSONObject arrayObject = array.getJSONObject(i);
7                          if (arrayObject.getString("id").equals("
                             humidity")){
8                            String value1 = arrayObject.getString("
                               current_value");
9                            textView3.setText(value1);
10                         }
11                         if (arrayObject.getString("id").equals("
                             temperature")){
12                           String value2 = arrayObject.getString("
                               current_value");
13                           textView5.setText(value2);
14                           showToast("Success!");
15                         }
```

Výpis kódu 2.2: Parsing získaného objektu ve formátu JSON

Je také nutno modifikovat soubor **Android_manifest.xml**, který je jedním z nejdůležitějších souborů v aplikaci pro Android. Plný výpis tohoto souboru lze najít v příloze A.2. V tomto souboru jsou definovány všechny třídy v aplikaci a také název aplikace. Dále je zde definována tzv. „SDK verze“. Tato verze má hodnotu typu integer a informuje o tom, jakou minimální verzi operačního systému Android tato aplikace podporuje a na jakou verzi je aplikace vyvíjena[2]. Dále jsou zde definovány povolení která aplikace potřebuje ke své činnosti. Tato povolení jsou pro chod aplikace nezbytná a jsou mimo jiné i definována u každé aplikace na virtuálním obchodě Google Play. Pro správný chod aplikace je potřeba mít povolený přístup k internetu a k přihlašovacím údajům uživatele. Dále je u každé aktivity definován také název, který se zobrazuje při spuštění dané aktivity. Zde si můžeme všimnout, že zde není přesně definován text, ale je na něj pouze odkázáno pomocí operátoru „@“. Ten se odkazuje na soubor **res/values/strings.xml**, kde by správně měly být definovány

[2]Velikost proměnné typu integer je opět ovdozena od API Levelu.

všechny proměnné typu String v aplikaci. Je to z toho důvodu, že když je vyvíjena aplikace, která má být ve více jazycích, tak aby nemusel být veškerý text pracně přepisován do jazyka, který je požadován, stačí pouze přidat příponu k souboru values definovanou pomocí ISO kódu dané země. Takže pokud chceme přepsat text například do španělštiny, stačí pouze k souboru values přidat příponu -es a systém android se už postará o překlad sám. Jednoduchý „export“ řetězce (Stringu) do souboru **res/values/strings.xml** je tedy možný, pokud v XML souboru najedeme myší na daný řetězec, stiskneme kombinaci kláves ctrl + 1.

Tímto byla tedy vytvořena základní aplikace pro zobrazení dat v jednom virtuálním uzlu.

2.2 Vývoj finální verze semestrálního projektu

Aplikaci bylo také nutno upravit tak, aby bylo možné zobrazit několik uzlů a uložené hodnoty v nich. Prvním krokem tedy bylo vytvoření několika uzlů v aplikaci Outdoorview a dále je rozdělit do sítí. Rozdělení do sítí bylo dosaženo tak, že všem uzlům náležícím do dané sítě v byly položce „tags“ v metadata (viz strana č.10) pozměněny parametry. Uzly tedy byly rozděleny do sítí s názvy „Network1“ a „Network2“.

Dále bylo potřeba zjistit, jakým HTTP příkazem by se daly dané uzly zobrazit. Původní záměr byl provést zobrazení tak, že na dané URL adrese by byly zobrazeny všechny uzly spadající pod daný Master Key a poté by z pole „tags“ byly rozděleny do sítí. Bohužel na stránkách Xively daný příkaz není. Bylo možné pouze zobrazit uzly veřejné, které spadaly pod danou síť, uzly privátní však ne. Poté byl však nalezen příkaz, kterým se dají zobrazit všechny uzly náležící uživateli, který je vytvořil. Tento příkaz má tvar „https://api.xively.com/v2/feeds.json?user=xsedla98“. Tím se zobrazí uzly vytvořené pod účtem xsedla98. Tato možnost je však pro uživatele spíše příjemnější, protože aplikace může opět použít přihlašovací údaje zadané již na začátku aplikace a uživatel tedy nemusí zbytečně vypisovat dlouhý a těžko zapamatovatelný Master Key.

Tato nová aktivita, která zobrazuje seznam uzlů nese název **NodeSelection** a je provázána s grafickým souborem **node_selection.xml**. Zobrazení vypadá tak, že do textového pole uživatel zadá název sítě, kterou chce zobrazit a v tzv. „ListView“ se mu zobrazí seznam všech uzlů v dané síti. Podrobný popis obejktu ListView je poměrně obsáhlý, jeho detailní popis lze najít v [1]. Zjednodušeně řečeno je ListView seznam proměnné délky, do kterého je možné vypisovat hodnoty. ListView čte hodnoty buď z pole řetězců anebo z Javovské kolekce s názvem List. Zde byl pro ukládání hodnot zvolen tzv „ArrayList“, což je dynamické pole, do kterého lze hodnoty ukládat. Dynamické znamená, že jeho délka je proměnná. Předání těchto

parametrů objektu ListView je prováděno za pomocí tzv. „Adaptéru“. Základní dělení Adaptéru je podle typu objektu, ze kterého data získává. Upřesnění jeho funkce a typů lze opět najít v [1]. Pro aplikaci byl tedy zvolen tzv. „ArrayAdapter“, který pro čtení využívá právě pole. ListView tedy využívá tento Adaptér k naplnění sebe sama hodnotami.

Logika programu je napsána tak, aby první hodnota v poli „tags“ byla porovnávána s textem, který uživatel zadává v textovém poli. Pokud je nalezena shoda, tedy pokud je zadaná síť napsána správně, ListView se naplní hodnotami. Tyto hodnoty jsou všechny uzly, které do dané sítě náleží. Dále je v ListView nastaven tzv. „onItemClickListener“, který „naslouchá“, zda uživatel kliknul na ListView. V této části je nutno definovat, jak daný Listener bude vyhodnocovat místo kliknutí. Je možné toto místo specifikovat buď pomocí pozice nebo pomocí ID řádku. Byla zvolena první možnost, která řádky rozděluje podle indexu typu Integer. V ListView se tedy vypíše seznam uzlů pro danou síť, avšak tento seznam se vždy mění v závislosti na vybrané síti. Uzly jsou však vždy v sestupném pořadí neboť takto jsou brány z JSON Objektu, kde probíhá parsing. Do proměnné selected se ukládá text v řádku, na který uživatel právě kliknul v podobě řetězce. Tento řetězec je poté porovnáván s názvy jednotlivých uzlů, čímž se identifikuje ten správný. Listener poté naslouchá, na který řádek uživatel klikne a porovnává proměnnou seleceted s názvy uzlů. Jakmile najde ten správný, opět aplikace přejde pomocí Intentu do Aktivity Request, kde se zobrazí data pro daný uzel.

V tomto místě se vyskytl problém. A to jak použít správný HTTP příkaz pro daný uzel. Jak již bylo výše řečeno, každý uzel má definovanou svoji HTTP adresu, ve které jsou uložena data. HTTP adresa by se tedy v Aktivitě Request měla dynamicky měnit podle toho, na jaký uzel v Aktivitě NodeSelection uživatel klikne. Tento problém nakonec opět pomohl vyřešit Intent. Pomocí něj lze totiž mezi Aktivitami zasílat jednoduchá data. Takže daný Intent byl nastaven tak, že pokud uživatel klikne na nějaký uzel v ListView, předá Aktivita NodeSelection pomocí Intentu URL adresu náležící danému uzlu Aktivitě Request, která jej poté zpracuje a předá jej jako parametr pro třídu HttpComm. Toto řešení je velice jednoduché a efektivní, jinak by totiž každý uzel musel mít definovanou vlastní třídu a Aktivitu, což by bylo nesmyslně náročné.

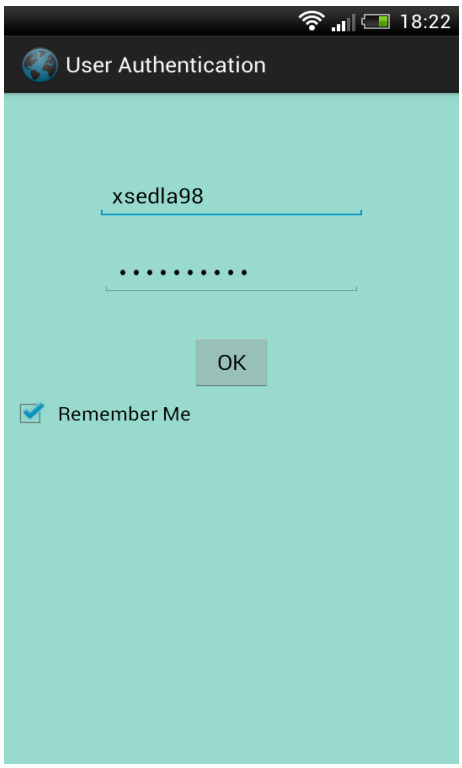
3 ROZŠÍŘENÍ ZÁKLADNÍ APLIKACE

V předchozí kapitole byl popsán vývoj základní verze aplikace, tedy výstup semestrálního projektu. Aplikace uměla provést autorizaci uživatele a na základě těchto dat provádět komunikaci s Xively. Uměla také zobrazit uzly náležící do stejné sítě a po kliknutí na jeden z nich zobrazit aktuální hodnoty v tabulce.

3.1 Pokračování ve vývoji

Dalším úkolem bylo aplikaci rozšířit tak, aby uživatel měl mnohem větší možnosti co se týče vyhledávání. Mělo by mu být umožněno kromě vyhledávání pomocí sítí také hledat pomocí názvu konkrétního uzlu a mít možnost vidět veřejné uzly ostatních uživatelů. Také by měl mít možnost vyhledávat pomocí hodnot v uzlech a konkrétních datastreamech a to podle rovnosti či velikosti. Dále by měl také mít možnost zvolit si tzv. „čerstvost dat“, tedy například nechce zobrazit data, která jsou starší než jedna hodina.

Původní část aplikace tedy zůstává nezměněna, uživatel je opět přihlášen a aplikace ověřuje správnost údajů a zda je zaškrtnutý CheckBox. Výsledná podoba této Aktivity je na obrázku č.3.1.



Obr. 3.1: Úvodní obrazovka aplikace

Nově je zde implementována metoda **connectivityTest()**, která kontroluje, zda je uživatel připojen k síti Internet. Tato metoda je důležitá, neboť při komunikaci se volá třída HttpComm, která poté do volající třídy vrací výsledek. Avšak při vypnutém připojení třída vrátí parametr **null** a ve volající třídě tedy nastane výjimka **NullPointerException**, která způsobí pád aplikace. I když bude tato výjimka ošetřena, k pádu aplikace sice nedojde, avšak uživatel se nedozví, že k nějaké chybě došlo. Metoda kontrolující připojení je vypsána ve výpise č.3.1.

```
1 public boolean connectivityTest() {
2         ConnectivityManager cm = (ConnectivityManager) getSystemService
              (Context.CONNECTIVITY_SERVICE);
3         NetworkInfo networkInfo = cm.getActiveNetworkInfo();
4         if (networkInfo != null && networkInfo.isConnectedOrConnecting
              ()) {
5             return true;
6         }
7         else
8         return false;
9     }
```

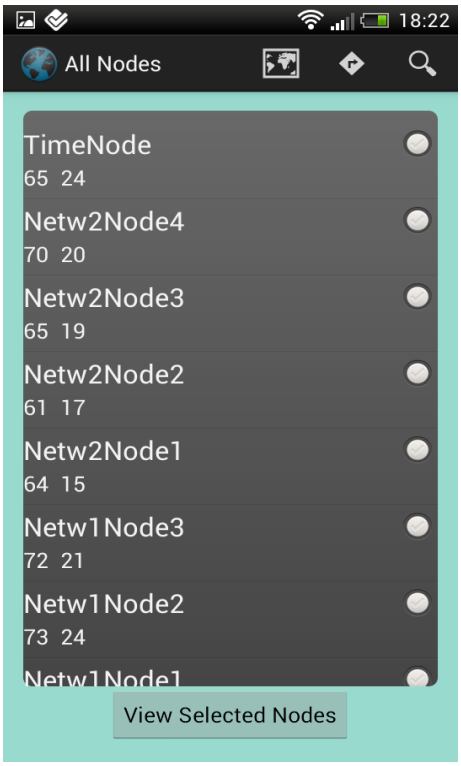
Výpis kódu 3.1: Kontrola připojení k síti Internet

Metoda tedy vrací parametr **false**, pokud připojení není uskutečněno. Po požadavku na zahájení komunikace v dané třídě je tedy zavolána instance třídy HttpComm a ta vrátí nějaký výsledek. Pokud je výsledkem parametr null, aplikace nebude ukončena neboť je tato výjimka ošetřena. Hned pod volání této instance je prováděna kontrola připojení. Pokud připojení není navázáno, tedy metoda connectivityTest() vrací parametr false, je uživatel vyzván krátkou zprávou ve formě Toastu k zapnutí internetového připojení v zařízení. Tuto metodu obsahují všechny Aktivity aplikace, které provádí síťovou komunikaci. Kontrola je prováděna stejným způsobem, jako bylo popsáno výše.

Dále je v této třídě také použita metoda **onBackPressed()**, což je jedna z metod, kterou implementuje každá Aktivita. Je zde použita anotace **@Override**, čímž dojde k tzv. „překrytí“ původní metody. Metoda onBackPressed() souvisí se stisknutím tlačítka „zpět“ na hardwarové klávesnici telefonu. Pokud by metoda byla nezměněna, došlo by vždy k návratu do předchozí Aktivity. Tedy pokud bychom chtěli z aplikace vyjet pomocí několika stisknutí tlačítka zpět, dostali bychom se do této původní Aktivity (tedy na úvodní obrazovku aplikace) a po dalším stisknutí tlačítka zpět opět do Aktivity předchozí a tak by tedy došlo k zacyklení a uživatel by neměl možnost z aplikace vyjet jinak, než pomocí stisku tlačítka „domů“. Tato přepsaná metoda funguje tak, že se po stisku tlačítka zpět v této aktivitě objeví zpráva ve formě Toastu, která uživatele informuje, že po dalším stisku tlačítka zpět dojde k opuštění aplikace. Pokud stiskne tlačítko zpět do dvou sekund od prvního

stisknutí, dojde k přepnutí na hlavní obrazovku telefonu. Pokud by časový interval mezi stisknutími byl větší než dvě sekundy, došlo by po dalším stisku tlačítka opět k vypsání původní zprávy. Uživatel sice z aplikace vyjede, avšak ta stále běží na pozadí. Správně by totiž aplikace nikdy neměla být ukončena uživatelem a správa aplikací by měla být čistě pod správou operačního systému. Až ten sám rozhoduje podle dostupného místa v paměti, zda aplikaci ponechávat v paměti, či ji z ní kvůli nedostatku uvolnit. Tím je způsobeno to, že když v systému Android někdy vyjedeme z aplikace stiskem tlačítka „domů“ a poté se do aplikace vrátíme (třeba i druhý den), aplikace opět běží ve stejné Aktivitě, v jaké jsme aplikaci posledně opustili, neboť systém aplikaci stále uchoval v paměti.

Pokud jou tedy zadané údaje správné a je zaškrtnutý CheckBox, aplikace přejde do další Aktivity s názvem **Node View**. Tato Aktivita je uvedena na obrázku č.3.2.

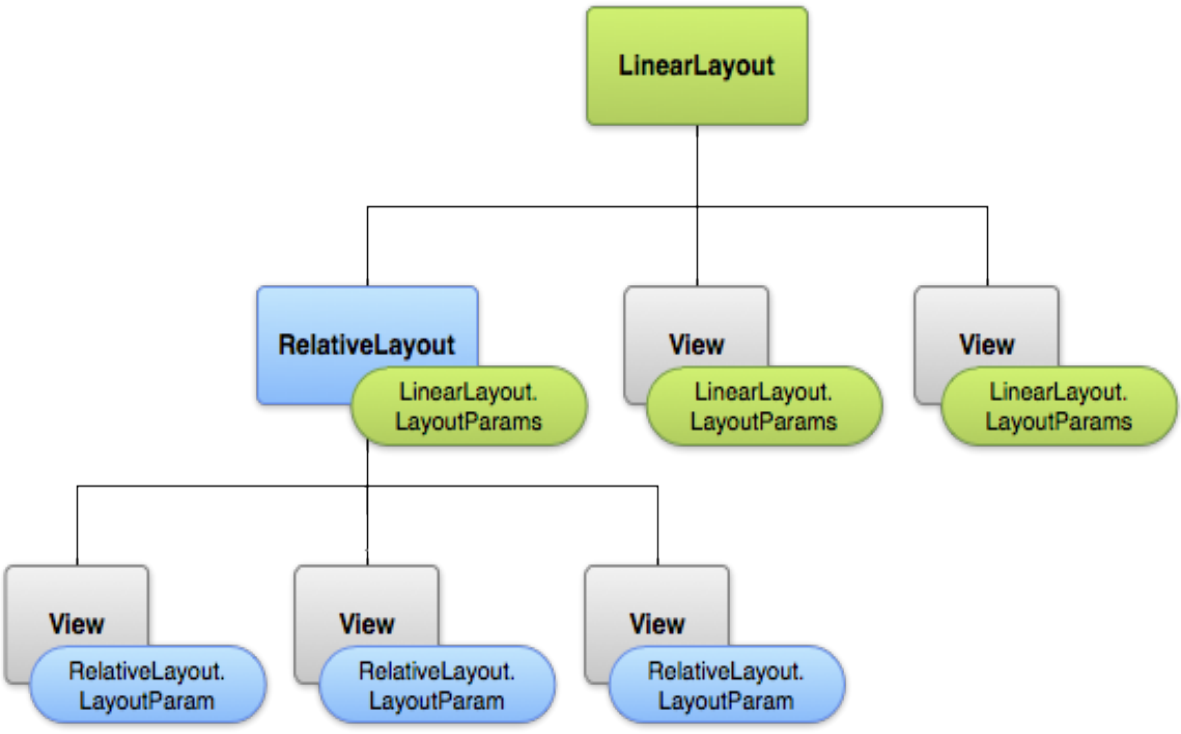


Obr. 3.2: Zobrazení všech uzlů v Aktivitě Node View

Zde vidí uživatel v objektu ListView seznam všech jím vytvořených uzlů. Vedle názvu uzlů se v seznamu nachází CheckBoxy, jejichž označením uživatel dává najevo, které uzly chce zobrazit. Zároveň jsou pod názvy uzlů uvedeny hodnoty datastreamů jednotlivých uzlů, uživatel má tedy možnost ihned po přihlášení do aplikace vidět okamžité hodnoty v jednotlivých uzlech. Tato funkce může být kritická hlavně u těch uzlů, kde uživatel potřebuje často monitorovat stav jednotlivých datastreamů. Pod tímto seznamem se také nachází tláčítko „View Selected Nodes“. Pokud uži-

vatel vybere jeden uzel a stiskne tlačítko, dostane se do další Aktivity s názvem **Datastream Selection**. Zde uvidí v seznamu pod sebou jednotlivé datastreamy náležící vybranému uzlu a vedle nich také jejich hodnoty. Pokud by v Aktivitě Node View vybral více uzlů a stiskl tlačítko, byl by přesměrován do třídy s názvem **All Graph View**, kde se mu pod sebou zobrazí v grafech hodnoty všech datastreamů všech vybraných uzlů rozlišených pomocí legendy grafu.

Aby však mohl objekt ListView obsahovat tolik prvků, je nutné vytvořit si vlastní objekt pro jednotlivé řádky, neboť Android v základu nabízí pouze jednoduché objekty obsahující pouze buď jedno textové pole nebo textové pole a CheckBox. Zde bylo však nutno vytvořit Jedno textové pole pro název uzlů, dále CheckBoxy pro indikaci výběru uzlu a dále několik textových polí pro zobrazení hodnot jednotlivých datastreamů. Z tohoto důvodu byl vytvořen soubor s názvem **row2.xml**. Při vytváření grafických souborů v jazyce XML je nutné hned na začátku definovat, do jakého „layoutu“ budou jednotlivé objekty vkládány. Struktura těchto layoutů je uvedena na obrázku č.3.3.



Obr. 3.3: Struktura layoutů (převzato z [1])

Z hierarchie je patrné, že všechny layouty vychází ze základního LinearLayout. V každém souboru XML je však definován tzv. „root element“, který „drží“ jednotlivé objekty(„children“). Root element může být typu LinearLayout, RelativeLayout nebo Web View. První z nich řadí jednotlivé objekty vždy do jedné horizontální či

vertikální řady pod sebou. Naproti tomu RelativeLayout řadí své objekty do relativních pozic. To znamená, že je teoreticky možné je umístit kamkoliv na zvolenou obrazovku. Poslední z nich, Web View, umožňuje v dané Aktivitě zobrazit webové stránky [1].

Pro vytvoření řádku v ListView byl tedy použit RelativeLayout, neboť je zde nutné prvky uspořádat trochu složitějším způsobem. Prvek, který se nachází v řádku hned nahoře nese název „CheckedTextView“. Jak z názvu vyplývá, tento objekt v sobě kombinuje funkci jednoduchého textového pole a CheckBoxu. Do textového pole tohoto objektu jsou vypisovány názvy jednotlivých uzlů a pokud dojde ke kliknutí na řádek, je tento objekt označen. Dále se v řádku nachází několik textových polí neboli TextView, do kterých jsou ukládány hodnoty jednotlivých datastreamů.

Naplnění takto vytvořených řádků objektu ListView hodnotami již není tak triviální záležitost. Je totiž nutné definovat, do kterých objektů se mají dané hodnoty ukládat. Pro tento účel je zde použit tzv. „SimpleAdapter“, jenž je uveden ve výpise č.3.2. Pro přiřazování hodnot objektům je použita Javovská kolekce s názvem HashMap. Struktura adaptéru je následující: V první části je uveden kontext, tedy kde se objekt nachází. Dále je uveden zdroj dat, což musí být ArrayList, který implementuje kolekci HashMap. Poté je uveden grafický soubor, který má adaptér použít (row2) a nakonec jsou již pomocí polí typu String a integer definovány jednotlivé objekty, do kterých se data vkládají. Takže pokud je například do mapy přidána proměnná s klíčem „title“ a tato mapa je poté implementována kolekcí list4, kterou Adaptér používá k zápisu dat, dojde k naplnění objektu text2 (zde CheckedTextView) hodnotami. Z výpisu je také vidět, že pro zápis dat je definováno pět textových polí, tedy je možné do seznamu načíst až pět prvních datastreamů, avšak tato velikost může být kdykoliv jednoduše rozšířena. Pokud k výpisu používáme například jen dvě pole, je zbytek nevyužitých polí prázdný. „Fyzicky“ se tedy pole pořád v řádku nachází, avšak pro uživatele nejsou viditelné.

```
1 SimpleAdapter adapter = new SimpleAdapter(getApplicationContext(),
    list4, R.layout.row2,
2    new String[] {"title", "value1", "value2", "value3", "value4",
3    "value5"},
4    new int[] { android.R.id.text2, android.R.id.text1, R.id.textView1,
5    R.id.textView2, R.id.textView3, R.id.textView4});
```

Výpis kódu 3.2: Adaptér typu SimpleAdapter

Zároveň jsou v této Aktivitě (i ve všech ostatních aktivitách) implementována tlačítka na horní liště aplikace. Jejich cílem je především usnadnění navigace v aplikaci. Stiskem levého tlačítka s ikonou mapy se uživatel dostane do tzv. „Outdoor View“, což je zobrazení všech jeho uzlů na mapě. Po stisku tohoto tlačítka aplikace spustí webový prohlížeč, ve kterém jsou uzly zobrazeny. Pro zobrazení byla

opět použita aplikace Outdoorview vytvořená týmem Wislab. Do HTML příkazu se automaticky doplňuje uživatelské jméno podle zadané hodnoty v první Aktivitě a zobrazují se tedy jen uzly patřící danému uživateli. Dalším prvkem v horní liště je tlačítko „Set Filter“ s ikonou lupy. Jeho cílem je přepnutí do Aktivity, jenž má na starost vyhledávání uzlů. O této této funkcionalitě bude pojednáno později. Prostřední tlačítko s ikonou cedule je zde zavedeno pro rychlý přechod zpět. Z jakékoliv Aktivity lze pomocí toho tlačítka rychle přepnout na Aktivitu Node View. Názvy jednotlivých tlačítek se uživateli zobrazí po jejich dlouhém stisknutí.

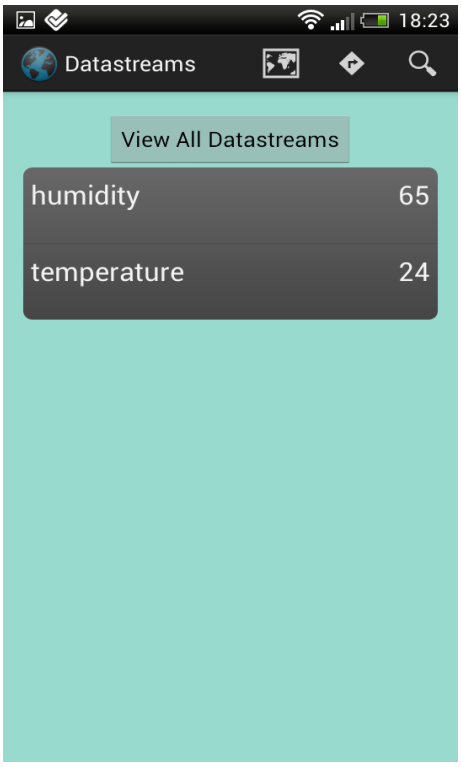
Tyto tlačítka se v jednotlivých aktivitách implementují pomocí tzv. „menu“. Při vytvoření nové aktivity dojde automaticky k vytvoření dvou XML souborů. Jeden z nich implementuje grafické rozhraní Aktivity, jak bylo popsáno výše a druhý z nich vytváří menu pro danou Aktivitu. Každá aktivita však svoje menu nutně mít nemusí. V základě každá Aktivita překrývá metodu **onCreateOptionsMenu()**, které Aktivitě menu vytváří. Původně menu každé Aktivity obsahuje pouze položku „Settings“. Vyváření jednotlivých položek menu tedy opět probíhá v souboru XML. Jsou zde definovány názvy jednotlivých položek a také jejich ikony. Překrývaná metoda **onOptionsItemSelected()** poté pomocí příkazu Switch Case reaguje na kliknutí na danou ikonu. Do horní lišty se vejdou tři takto vytvořené ikony, zbytek je přesunut do dalšího menu. Na starších telefonech se toto menu zobrazovalo pomocí stisku hardwarového tlačítka „menu“. Od verze Android 3.0 (API Level 11) však telefony toto tlačítko nevyužívají a místo toho používají tzv. „Three Dots Bar“, tedy softwarové tlačítko s ikonou obsahující tři tečky. Po stisku tohoto tlačítka se pod ním v seznamu objeví jednotlivé položky, které se do lišty již nevešly. Do tohoto menu Android při přeplnění lišty automaticky přesouvá všechny položky, které nemají explicitně nastaveno permanentní zobrazování. V této aplikaci však toto další menu není nutné použít, neboť lišta nikdy neobsahuje více, než tři položky.

3.2 Zobrazení dat v uzlech

V předchozí kapitole bylo řečeno, že po výběru uzlů ze seznamu a stisku tlačítka dojde k vizualizaci jejich dat. Tato vizualizace může být dvojího způsobu. Buď dojde k výpisu dat uložených v jednom uzlu v tabulce (seznamu) anebo ke zobrazení grafů všech datastreamů vybraných uzlů. Nejprve bude rozebrán první způsob.

Po výběru jednoho uzlu a stisku tlačítka tedy aplikace přejde do Aktivity **Datastream Selection**. Zde je zobrazen seznam všech datastreamů v daném ulzu a jejich současná hodnota. Výsledná podoba této Aktivity je uvedena na obrázku č.3.4.

Pokud zde uživatel stiskne některý z datastreamů, dostane se do další aktivity s názvem **Graph View**. Zde uvidí v grafu historické hodnoty uložené v daném

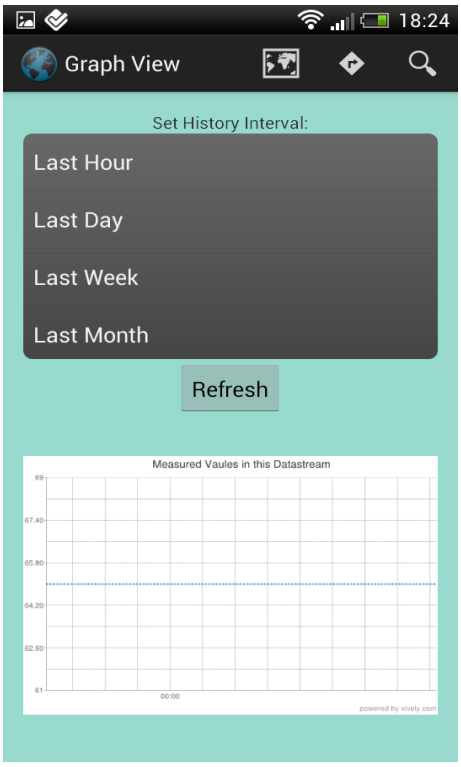


Obr. 3.4: Aktivita Datastream Selection

datastreamu. Pokud by stiskl tlačítko „View All Datastreams“, zobrazí se mu v grafu všechny datastreamy v tomto uzlu. Jednotlivé grafy jsou od sebe opět odlišeny pomocí názvu.

Zobrazení jednoho grafu je tedy poměrně jednoduché a je prováděno třídou **GraphView**. Aktivita Graph View využívající třídu GraphView je ukázána na obrázku č.3.5.

Výsledný graf je ve formě bitmapy, jenž je stažena přímo z Xively ve formátu png. Pro zobrazení grafu stačí pouze zadat API Endpoint daného uzlu a k němu přidat název vybraného datastreamu a příponu .png. Výsledný příkaz má tedy tento formát: **https://api.xively.com/v2/feeds/FEED_ID_HERE/ datastreams/DATASTREAM_ID.png**. Tímto však dojde pouze ke zobrazení základního grafu, který ani nemá časové měřítko, což je ve výsledku dosti málo informativní, obzvláště u uzlů, kdy je kritické sledovat chování jejich datastreamů v nějakém časovém horizontu. Naštěstí umožňuje Xively API nastavení několika dalších parametrů v grafu, jako je šířka či výška, barva, název a hlavně detailní zobrazení mřížky grafu a časová osa. Tyto parametry, vyjma časovou osu, jsou tedy v aplikaci nastaveny pevně, aby došlo k co nejdetailnějšímu zobrazení grafu. V základním nastavení Xively zobrazuje velikost dat za poslední den, tedy ihned po spuštění této Aktivity uživatel vidí hodnoty ve zvoleném datastreamu v grafu za uplynulý den.



Obr. 3.5: Aktivita Graph View

Nad grafem se nachází seznam časových intervalů, z nichž je možné si jeden zvolit. Původně byla tato volba vyřešena pomocí zadání hodnoty do textového pole, avšak uživatelsky je tato možnost celkem nepříznivá a zároveň by zadaná hodnota musela odpovídat formátu, který Xively podporuje[1]. Uživatel má tedy pevně nastavených několik časových intervalů, ze kterých si vybírá. Jsou zde použity stejné časové intervaly, jako v aplikaci Outdoorview, aby byla zachována jednotnost. Volba je navíc indikována pomocí zprávy, aby byl uživatel ujištěn, zda vybral správný interval. Po stiku tlačítka „Refresh“ se tedy automaticky změní časová osa grafu podle zvolené hodnoty. Pokud není žádný interval zadán, graf se zobrazuje ve výchozím měřítku.

Obdobným způsobem funguje i Aktivita **All Graph View**. Jejím úkolem, jak z názvu vyplývá, je zobrazit všechny zvolené grafy. Do této Aktivity tedy přechází Aktivity Node View a Datastream Selection. V této Aktivitě je jako root element nastavený tzv. „ScrollView“. Z názvu je patrné, že jde o zobrazení, které dynamicky rozšiřuje spodní část obrazovky podle počtu vytvořených objektů. V této Aktivitě totiž není předem jasné, kolik grafů se zde může objevit. Dalším velkým rozdílem oproti Aktivě Graph View spočívá v dynamickém vytváření objektů. V Aktivitě Graph View se nachází staticky vždy jen jeden graf a proto je možné jej jednoduše

[1]Formáty jsou v tomto tvaru: 1minute(s), 1hour(s) atd.

definovat v souboru XML. Zde se však grafy musí dynamicky vytvářet[2]. Vytváření jednotlivých grafů probíhá na základě získaných dat z Intentu. Intent této Aktivitě předává ArrayList, který obsahuje odkazy na jednotlivé grafy datastreamů podle vybraných prvků v předchozí aktivitě. Je tedy vytvořeno tolik grafů, kolik je v Intentu uloženo odkazů.

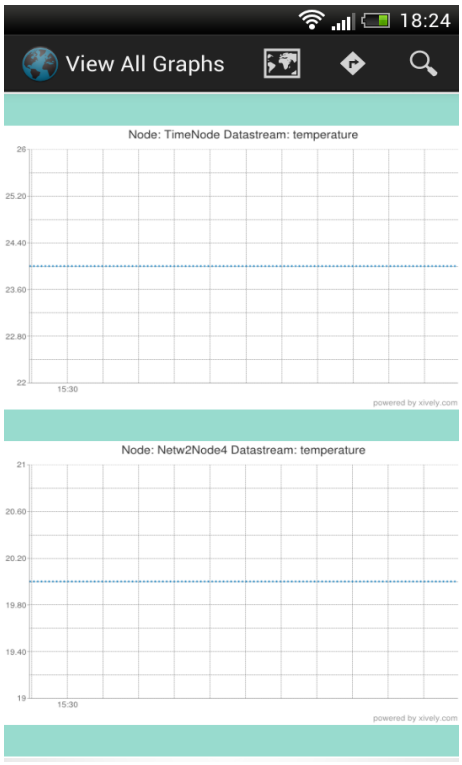
Po spuštění této Aktivity je tedy nejprve nutno zadat časový interval. Tato volba je stejná, jako v Aktivitě Graph View. Po stisku tlačítka je otevřeno nové okno, kde jsou zobrazeny všechny grafy se zvoleným časovým měřítkem. Tato Aktivita je zdaleka nejnáročnější na výpočetní výkon zařízení, neboť pro zobrazení jednotlivých grafů je nutné provést opakovaně síťovou komunikaci a protože jsou grafy vytvářeny dynamicky, zařízení musí pro každý z grafů alokovat místo v paměti. Zobrazení tedy i na rychlém zařízení chvíli trvá. Pro změnu časového měřítka je nutno stisknout tlačítko zpět a opět zvolit nový časový interval. Uvedený postup je tedy zopakován, avšak nejdříve je ještě také nutné staré grafy z layoutu vymazat. Jak již bylo výše zmíněno, jednotlivé grafy jsou od sebe odlišeny pomocí názvu. Pokud je navíc tato třída AllGraphView spuštěna z Aktivity Node View, jsou navíc do grafu přidány i názvy jednotlivých uzlů, aby byly odlišeny jak uzly tak datastreamy. Podoba Aktivity včetně demonstrace rozlišení jednotlivých grafů je na obrázku č.3.6.

Pro komunikaci s Xively a následné zobrazení grafů se používá třída **GraphComm**. Původní třídu pro komunikaci HttpComm zde již není možné použít, neboť návratová hodnota této třídy je typu JSONObject a pro zobrazení grafů je potřeba jako návratová hodnota bitmapa a JSONObject není možné na bitmapu přetypovat. Výpis třídy GraphComm je tedy obdobný jako u třídy HttpComm (příloha č.A.1). Odpověď od serveru je nejprve přetypována do binárního vstupního proudu InputStream, který pracuje přímo se samotnými bajty. Tento proud bajtů je potom dekódován do podoby bitmapy pomocí třídy BitmapFactory. Tento výsledek je poté opět vrácen třídě, která instanci GraphComm zavolala.

U každého grafu je také možnost exportu dat. Export spočívá v kliknutí na zvolený graf. Po dlouhém kliknutí se zavolá metoda **saveImage()**, jenž zobrazí dialog, který se uživatele táže, zda chce opravdu daný graf uložit. Pokud zvolí tlačítko „yes“, metoda se pokusí daný obrázek uložit na SD kartu telefonu do galerie obrázků. Výsledný obrázek je uložen v maximální možné kvalitě ve formátu png. O výsledku ukládání je uživatel informován pomocí krátké zprávy. U ukládání je také nutno definovat název obrázku. Pokud byl název uveden staticky, došlo by pokaždé k uložení obrázku, avšak obrázek uložený předtím by byl přepsán. Proto je při ukládání použita Javovská třída UUID[3], která je vlastně pseudo - náhodný generátor, který

[2]V Androidu lze objekty vytvářet buď staticky pomocí jazyka XML nebo dynamicky za běhu („in Runtime“).

[3]UUID = Universally Unique Identifier



Obr. 3.6: Jednotlivé grafy v Aktivitě All Graph View

vždy vytvoří nový 128 - bitový název. Tímto tedy nedojde k vzájemnému přepsání obrázků. Pro povolení aplikace k zápisu do externí paměti dat je ještě nutno toto právo deklarovat v souboru AndroidManifest.xml (příloha č.A.2) pomocí parametru „WRITE_EXTERNAL_STORAGE“.

V obou Aktivitách dochází ke zobrazení grafů v maximální možné velikosti, tedy s parametry „MATCH_PARENT[4]“, čímž je objektu dána možnost využít největší potřebnou velikost pro svoje zobrazení. Přesto jsou však grafy poměrně malé a špatně čitelné, hlavně legenda grafu a číslování obou os. Pro lepší zobrazení byl tedy v obou Aktivitách zaveden zoom. Jelikož Android v základu nenabízí žádnou knihovnu pro implementaci přiblížení na zvolený objekt, je nutno si takovou knihovnu vytvořit, či stáhnout neoficiální verzi. V této aplikaci je použita volně stažitelná třída s názvem **TouchImageView**, kterou lze získat z následujících stránek: https://github.com/MikeOrtiz/TouchImageView. Zavedení třídy je poměrně jednoduché, stačí pouze deklarovat objekt, na který bude použito přiblížení do proměnné „TouchImageView“. Výsledný objekt tedy bude možné přiblížit buď dvojitým poklepáním či klasickým „Pinch Zoom“, tedy oddálením dvou prstů od

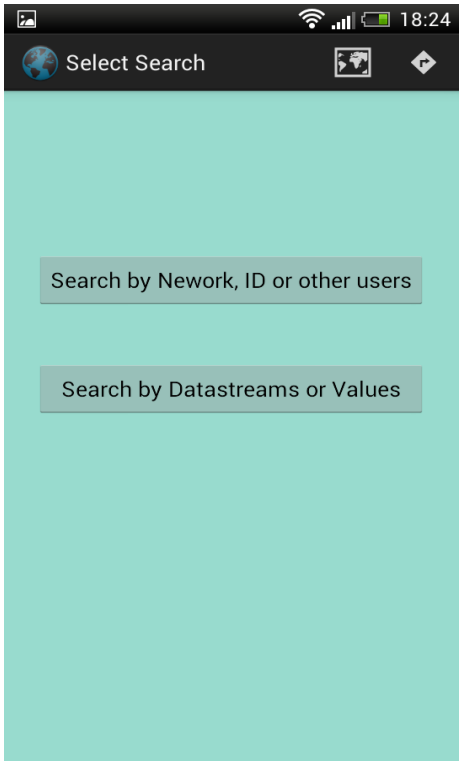
[4]Do verze API 8 se využíval parametr FILL_PARENT. Spousta programátorů jej doteď používá, avšak správně by se měl používat novější MATCH_PARENT.

sebe. Ve třídě se navíc dá do proměnných nastavit, jak veliké bude výsledné přiblížení.

3.3 Implementace vyhledávání

Dalším krokem ve vývoji aplikace po zavedení zobrazování dat, bylo jejich vyhledávání. Původní aplikace uměla po výstupu semestrálního projektu hledat uzly pouze podle sítě, do které náleží. Bylo tedy nutné možnosti v této oblasti rozšířit a dát uživateli více možností, co se týče hledání.

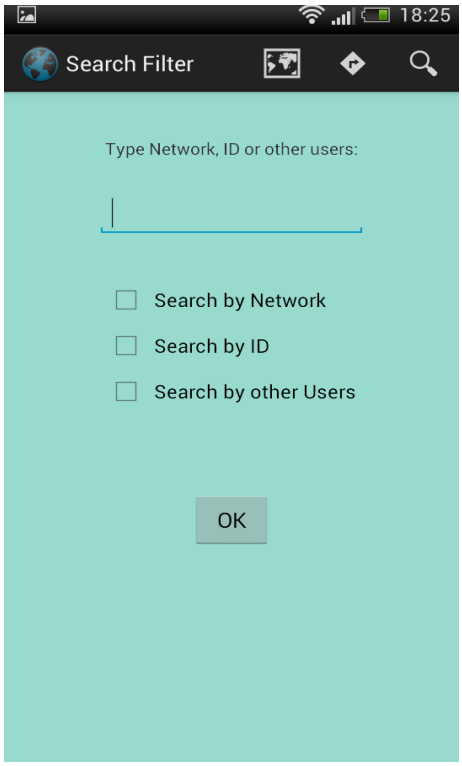
V každé Aktivitě v aplikaci, tedy kromě úvodní obrazovky, je v horní liště uvedeno tlačítko „Set Filter“ s ikonou lupy. Po stisku tohoto tlačítka je uživatel přesměrován do Aktivity s názvem **Search Selection** (obrázek č.3.7).



Obr. 3.7: Aktivita Search Selection

V této Aktivitě si dále vybírá, dle jakých kritérií chce vyhledávat. První možností je po stisknutí prvního tlačítka vyhledávat opět podle sítě, názvu nebo identifikátoru určitého ulzu anebo je zde také uvedena možnost vyhledávat uzly ostatních uživatelů. Po stisku prvního tlačítka tedy následuje Aktivita **Search Filter One**, která

obsahuje textové pole EditText, tři CheckBoxy a tlačítko „OK“. Vzhled Aktivity je uveden na obrázku č.3.8.

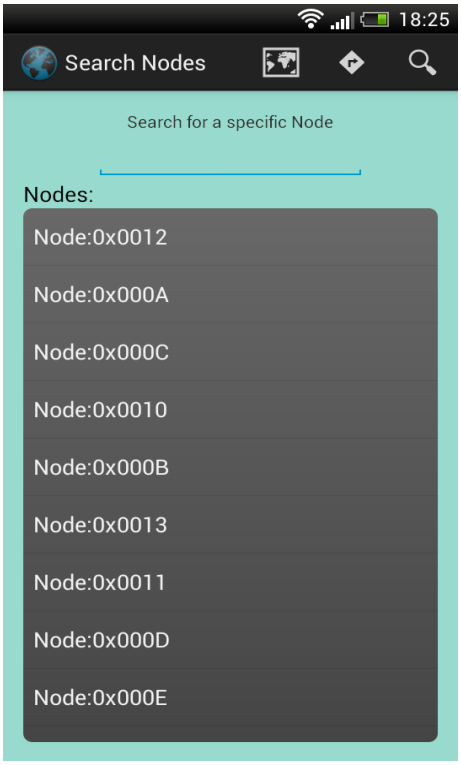


Obr. 3.8: Aktivita Search Filter One

Do textového pole zadá uživatel hodnotu, podle které chce vyhledávat a z CheckBoxů si vybere výše uvedená vyhledávací kritéria. Pokud tedy správně zadá název sítě, jsou mu v následující Aktivitě v seznamu zobrazeny všechny uzly, které splňují daná kritéria. Stejný postup je i u vyhledávání podle identifikátoru uzlu a podle ostatních uživatelů, avšak jak bylo již zmíněno, je možné vyhledávat pouze veřejné uzly jiných uživatelů. Pokud by byl text zadán špatně či žádné uzly daným kritériím neodpovídají, je o tom uživatel informován chybovou hláškou. Zároveň pokud není vybrán žádný CheckBox, tedy nejsou zvolena žádná vyhledávací kritéria, je na tuto skutečnost opět po stisku tlačítka upozorněno.

Pokud jsou tedy vyhledávací parametry zadány správně, uživatel uvidí v další Aktivitě v seznamu uzly odpovídající kritériím. Nad seznamem je ještě také textové, pomocí kterého je možné provést ještě dodatečné vyhledávání v seznamu. Kdyby totiž daným kritériím vyhovovalo například padesát uzlů, těžko bychom v takovém seznamu na malém telefonu hledali například jeden konkrétní uzel. Proto je zde uvedena tato možnost dodatečně filtrovat zobrazený seznam pomocí napsaného textu. Po kliknutí na zvolený uzel aplikace opět přejde do Aktivity Datastream Selection, kde je opět seznam všech datastreamů daného uzlu i s hodnotami. Výsledná podoba

této Aktivity je ukázána na obrázku č.3.9. Na obrázku jsou v seznamu vyfiltrovány uzly uživatele Wislab.

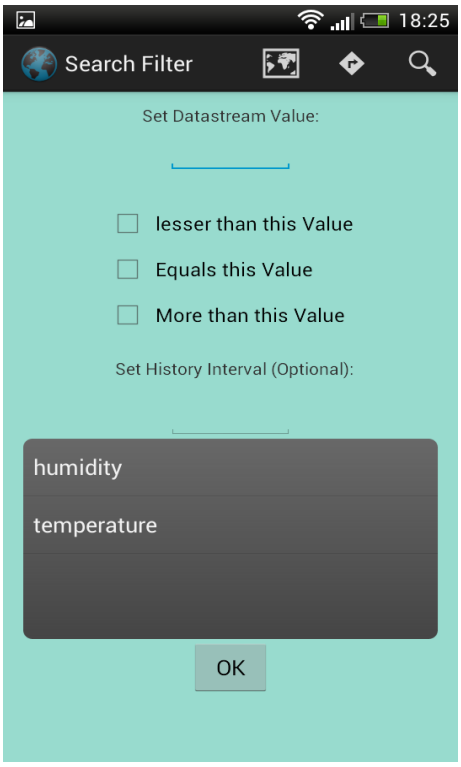


Obr. 3.9: Výpis uzlů podle zadaného uživatele

Pokud uživatel v Aktivitě Search Selection stiskne druhé tlačítko, dostane se do Aktivity s názvem **Search Filter Two**. V této aktivitě je možné vyhledávat konkrétní datastreamy a hodnoty v nich. Struktura Aktivity je na obrázku č.3.10.

Nahoře je opět pole EditText, do kterého je zadávána hodnota vyhledávaných datastreamů. Pod tímto polem jsou tři CheckBoxy, které představují operátory větší, menší, rovná se. Uživatel tedy zadá do textového pole hodnotu a dále si pomocí CheckBoxu vybere, zda chce zobrazit datastreamy, které mají hodnotu větší, menší nebo rovnou zadané hodnotě. Pod těmito CheckBoxy je dále další textové pole, do nějž uživatel může zadat tzv.„čerstvost dat“, tedy zadá časový interval (v minutách), který indikuje, že nechce zobrazit data, která jsou starší než tento interval. Použití tohoto pole je však nepovinné, pokud jej uživatel nechce použít, ponechá pole prázdné. Pod tímto polem se v seznamu nachází všechny datastreamy uzlů, ze kterých je možné si vybrat, který z nich bude pro vyhledávání použit (vždy lze použít pouzde jeden). Výběr datastreamu je opět indikován zprávou, aby bylo jasné, který ze seznamu byl zvolen.

Jak bylo v předchozím odstavci řečeno, použití časového intervalu je nepovinné. Pokud jej tedy uživatel nechce použít, ponechá pole prázdné. Pokud však uživatel



Obr. 3.10: Aktivita Search Filter Two

interval použít chce, stačí do pole zadat číslo typu integer. Pokud zadá neplatný formát čísla, je na to upozorněn pomocí chybové hlášky. Umožnění nastavení časového intervalu bylo poněkud obtížnější. Xively sice posílá tzv. „Timestamp“, neboli poslední změnu dat v datastreamu vyjádřenou v čase v parametru „at“, avšak zasílá jej ve formátu ISO8601, tedy „yyyy-mm-ddTHH:MM:SS.SSS“. Zároveň data ukládá a posílá v základním časovém pásmu UTC. Bylo tedy nutné nějakým způsobem získat současný čas a datum systému, ideálně také ve formátu ISO8601, a porovnat rozdíl tohoto času a času v parametru „at“ ve stejném časovém pásmu v minutách. Původní Javovské třídy sice umožňují získávat současnou hodnotu času systému, avšak pořád zůstává problém se synchronizací časových pásem a vyjádřením rozdílu v minutách. I kdyby by byl tento problém vyřešen tím, že budou jednoduše od současného času odečteny dvě hodiny a tím bude čas převeden do základního pásma, bude tato funkce fungovat pouze v pásmu UTC + 2 a navíc pouze v období letního času.

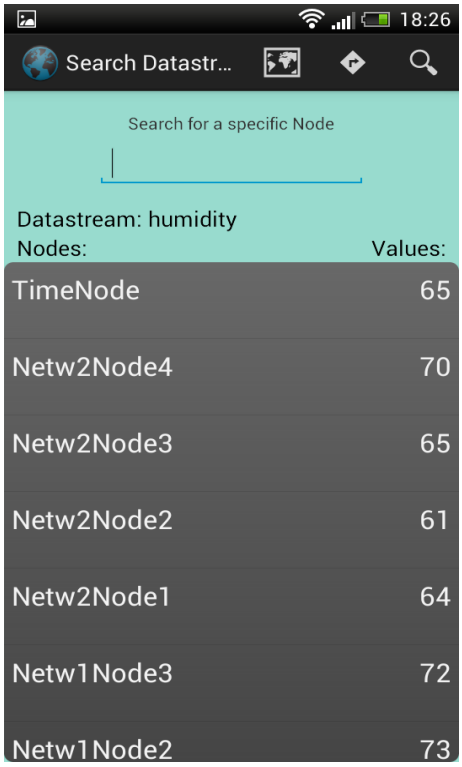
Tento problém nakonec pomohlo vyřešit rozšíření pro jazyk Java s názvem Joda Time. Toto rozšíření lze zdarma stáhnout na těchto stránkách: http://www.joda.org/joda-time/, kde je navíc dostupná dokumentace všech tříd a metod tohoto rozšíření včetně tutoriálů. Použití rozšíření v aplikaci je naznačeno ve výpise č.3.3. Nejprve se nastavuje časové pásmo, které chceme používat. Pokud je nastaveno

výchozí časové pásmo, jako v tomto případě, je jednoduše zjištěno časové pásmo, v němž se zařízení nachází. Poté je během komunikace s Xively zjištěn Timestamp zvoleného datastreamu (v pásmu UTC) a ten je poté převeden do stejného pásma, jako zařízení. Poté se vytvoří proměnná typu Interval, která porovnává rozdíly v obou časech. Do proměnné difference typu integer je potom ukládán rozdíl obou časů v minutách. Tato proměnná se poté snadno porovná s velikostí časového intervalu (taktéž typu integer), který byl uživatelem zadán v textovém poli.

```
1 DateTimeZone defaultZone = DateTimeZone.getDefault();
2 DateTime time = new DateTime(defaultZone);
3 DateTime feedTime = new DateTime(arrayObject1.getString("at").toString
      ());
4 Interval interval = new Interval(feedTime, time);
5 int difference = interval.toDuration().toPeriod().getHours()*60+
      interval.toDuration().toPeriod().getMinutes();
```

Výpis kódu 3.3: Časová synchronizace pomocí rozšíření Joda Time

Pokud jsou opět zadány všechny parametry správně, aplikace přejde po stisku tlačítka do nové Aktivity (obrázek č.3.11).



Obr. 3.11: Výpis jednotlivých datastreamů i s hodnotami

Pokud není vybrán žádný CheckBox, ale pouze datastream, zobrazí se hodnoty všech odpovídajících datastreamů všech uzlů. Při použití časového intervalu jsou

zobrazeny pouze datastreamy, které nemají data starší než zadaný interval. Pokud žádné takové datastreamy nejsou, dojde opět k výpisu chybové hlášky. V této Aktivitě se opět nachází textové pole typu EditText pro dodatečné vyhledávání v rámci seznamu. Dále je zde také textové pole, které informuje, o který datastream se jedná. A nakonec samotný seznam obsahující na jedné straně názvy uzlů, ve kterých se daný datastream nachází a na straně druhé současnou hodnotu v něm. Hodnota zadaná v textovém poli je porovnávána s kompletním textem v každém řádku v seznamu, takže je v tomto seznamu možno dodatečně hledat jak podle názvu uzlu, tak podle hodnoty datastreamu. Po kliknutí na zvolený řádek aplikace opět přejde do Aktivity Graph View, ve které je zobrazen graf s hodnotami zvoleného datastreamu.

3.4 Dokončení aplikace

Po splnění všech funkčních podmínek zbývala už jen grafická úprava aplikace, aby byla pro uživatele vizuálně příjemná. Došlo ke změně barvy pozadí z výchozí bílé barvy, změna grafických tvarů a barev jednotlivých objektů ListView a také byl upraven přechod mezi jednotlivými Aktivitami, aby aplikace působila plynuleji. Na závěr byla také vytvořena ikona k aplikaci pomocí této stránky. Na stránce lze ikonu jednoduše a rychle vytvořit a navíc je výsledná ikona uložena ve formátech mdpi - xxxhdpi, tedy od nejmenšího podporovaného formátu až po největší. Pro ikonu aplikace i všechny ostatní ikony a obrázky v aplikaci je použit formát hdpi, tedy formát pro zařízení s vysokým rozlišením. Tento formát bude mít dobré rozlišení jak na menších, tak i větších zařízeních.

Vizuální podoba aplikace je ukázána na předchozích obrázcích, které byly pořízeny pomocí screenshotu přímo z fyzického zařízení, na němž probíhal vývoj aplikace. Tímto byl tedy vývoj aplikace ukončen, neboť byly splněny všechny funkční požadavky zadání a zároveň byl vylepšen výsledný vzhled.

4 ZÁVĚR

Úkolem této práce bylo vytvořit aplikaci pro operační systém Android schopnou zobrazit uložená data na webovém cloudu. Aplikace je schopná připojit se na základě zadaných uživatelských údajů pomocí síťového protokolu HTTPS ke cloudu a stáhnout zvolená data. V úvodní části aplikace provádí autentizaci uživatele. Autentizaci ve skutečnosti provádí webový cloud a aplikace pouze na základě odpovědi vyhodnocuje, zda byly vloženy správné přihlašovací údaje. Tím je tedy vyřešena autentizace uživatele. V další části aplikace jsou vypsány všechny uzly s aktuálními naměřenými hodnotami, které jsou na cloudu uloženy pod daným uživatelským jménem. Zde si uživatel může vybrat, jakým způsobem si data zobrazí. Zobrazení je dvojího typu. Buď se data uložená v datastreamech opět zobrazí v seznamu se současnými hodnotami platnými pro vybraný uzel nebo je dále možno data zobrazit pomocí grafu a to pro libovolný počet uzlů a datastreamů, které jsou mezi sebou odlišeny pomocí legendy grafu. U grafů je dále umožněn zoom pro lepší čitelnost dat a zároveň také možnost uložení zvoleného grafu na externí paměť zařízení.

Dále je v aplikaci umožněno vyhledávání uzlů a datastreamů pomocí široké škály kritérií. Uživatel má tedy možnost zvolit vyhledávací kritéria, pomocí nichž se mu v seznamu vyfiltrují uzly či datastreamy splňující zadaná kritéria. Následné zobrazení dat probíhá stejně, jak bylo popsáno výše. U vyhledávání datastreamů je navíc uvedena možnost filtrovat data podle jejich stáří v minutách. Tato funkcionalita je synchronizována s časovým pásmem, ve kterém se telefon nachází a je tedy použitelná v jakémkoliv místě na Zemi.

Aplikace je vytvořena tak, že je možné ji použít v pro jakýkoliv uživatelský účet a pro jakýkoliv typ dat. Musí být dodržena pouze jednotná syntaxe komunikace s webovým cloudem a uživatel musí mít na cloudu vytvořený účet. Aplikace má tedy dostatečně univerzální využití. Její výsledná podoba také splňuje všechny funkční požadavky, jež jsou uvedeny v zadání bakalářské práce.

LITERATURA

[1] Android Developers. [online]. [cit. 2013-12-21]. Dostupné z URL: <http://developer.android.com>.

[2] BURD, Barry A. *Java for dummies [online].* 5th Edition. Indianapolis, Indiana: Wiley, c2011, xx, 404 p. [cit. 2014-05-23]. –For dummies. ISBN 978-0-470-37174-9.

[3] BURNETTE, Ed. *Hello, Android: introducing Google's mobile development platform [online].* 3rd Edition. Raleigh, N.C.: Pragmatic Bookshelf, c2010, xviii, 262 p. [cit. 2014-05-23]. Pragmatic programmers. ISBN 19-343-5656-5.

[4] BURTON, Michael a Donn FELKER. *Android application development for dummies.* 2nd Edition. Hoboken, NJ: John Wiley & Sons, Inc., 2012, 386 stran. ISBN 978-1-118-38710-8.

[5] ČÍKA, Petr a Martin ZUKAL. *Multimediální služby: počítačová cvičení.* Vyd. 1. Brno: Vysoké učení technické, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2013. 66 s. ISBN 978-80-214-4721-9.

[6] FARAHANI, Shahin. *ZigBee wireless networks and transcievers.*. Amsterdam: Elsevier ; Newnes, 2008. ISBN 978-0-7506-8393-7.

[7] STOJMENOVIC, Ivan. *Handbook of Sensor Networks.* Hoboken, NJ: Wiley., 2005, 531 stran. ISBN 04-716-8472-4.

[8] Xively – Public Cloud for the Internet of Things. Dostupné z URL: <https://xively.com/dev/docs/api/>.

SEZNAM PŘÍLOH

A PLNÉ ZNĚNÍ KÓDU NEJDŮLEŽITĚJŠÍCH ČÁSTÍ PROGRAMU

A.1 Úplný výpis třídy HttpComm

```
1
2
3  package com.example.sensor;
4
5  import java.net.URI;
6  import org.apache.http.Header;
7  import org.apache.http.HttpEntity;
8  import org.apache.http.HttpResponse;
9  import org.apache.http.auth.UsernamePasswordCredentials;
10 import org.apache.http.client.HttpClient;
11 import org.apache.http.client.methods.HttpGet;
12 import org.apache.http.impl.auth.BasicScheme;
13 import org.apache.http.impl.client.DefaultHttpClient;
14 import org.apache.http.util.EntityUtils;
15 import org.json.JSONObject;
16 import android.os.AsyncTask;
17 import android.util.Log;
18
19
20 public class HttpComm extends AsyncTask<String, Void, JSONObject>{
21
22
23         @Override
24         protected JSONObject doInBackground(String... params) {
25           // TODO Auto-generated method stub
26
27
28           try {
29
30             String userName = params[0];
31                    String passWord = params[1];
32                    String uri = params[2];
33                    HttpClient client = new DefaultHttpClient();
34
35             HttpGet request = new HttpGet();
36                 request.setURI(new URI(uri));
37
38                       UsernamePasswordCredentials credentials = new
                             UsernamePasswordCredentials(userName, passWord)
                             ;
```

```
39                      BasicScheme scheme = new BasicScheme();
40                      Header authorizationHeader = scheme.authenticate(
                            credentials, request);
41                      request.addHeader(authorizationHeader);
42
43
44                  HttpResponse responseGet = client.execute(request);
45                  HttpEntity resEntityGet = responseGet.getEntity();
46
47
48                   if (resEntityGet != null) {
49
50
51                      String response = EntityUtils.toString(resEntityGet
                            );
52                      JSONObject response1 = new JSONObject(response);
53
54                        Log.i("GET RESPONSE", response);
55
56                        return response1;
57
58                   }
59              } catch (Exception e) {
60                   e.printStackTrace();
61
62              }
63          return null;
64        }
65        @Override
66        public void onPostExecute(JSONObject result) {
67
68
69          }
70
71 }
```

A.2 Úplný výpis souboru AndroidManifest.xml

```
1
2 <?xml version="1.0" encoding="utf-8"?>
3 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
4     package="com.example.sensor"
5     android:versionCode="1"
6     android:versionName="1.0" >
7
```

```
8     <uses-sdk
9         android:minSdkVersion="14"
10        android:targetSdkVersion="18" />
11
12    <uses-permission android:name="android.permission.INTERNET" />
13    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
14    <uses-permission android:name="android.permission.USE_CREDENTIALS"
          />
15    <uses-permission android:name="android.permission.
          ACCESS_NETWORK_STATE" />
16    <uses-permission android:name="android.permission.
          WRITE_EXTERNAL_STORAGE" />
17
18    <application
19        android:allowBackup="true"
20        android:icon="@drawable/ic_launcher"
21        android:label="@string/app_name"
22        android:theme="@style/AppTheme" >
23        <activity
24            android:name="com.example.sensor.MainActivity"
25            android:label="@string/app_name"
26            android:windowSoftInputMode="stateHidden" >
27            <intent-filter>
28                <action android:name="android.intent.action.MAIN" />
29
30                <category android:name="android.intent.category.
                      LAUNCHER" />
31            </intent-filter>
32        </activity>
33        <activity
34            android:name="com.example.sensor.NodeView"
35            android:label="@string/title_activity_node_view" >
36        </activity>
37        <activity
38            android:name="com.example.sensor.ViewSelection"
39            android:label="@string/title_activity_view_selection" >
40        </activity>
41        <activity
42            android:name="com.example.sensor.DatastreamSelection"
43            android:label="@string/title_activity_datastream_selection"
44            android:windowSoftInputMode="stateHidden" >
45        </activity>
46        <activity
47            android:name="com.example.sensor.GraphView"
48            android:label="@string/title_activity_graph_view" >
49        </activity>
```

```
50          <activity
51              android:name="com.example.sensor.SearchFilterOne"
52              android:label="@string/title_activity_search_filter"
53              android:windowSoftInputMode="stateHidden" >
54          </activity>
55          <activity
56              android:name="com.example.sensor.AllGraphView"
57              android:label="@string/title_activity_all_graph_view" >
58          </activity>
59          <activity
60              android:name="com.example.sensor.SearchSelection"
61              android:label="@string/title_activity_search_selection" >
62          </activity>
63          <activity
64              android:name="com.example.sensor.SearchFilterTwo"
65              android:label="@string/title_activity_search_filter_two"
66              android:windowSoftInputMode="adjustPan"
67              >
68          </activity>
69          <activity
70              android:name="com.example.sensor.ListSelectionOne"
71              android:label="@string/title_activity_list_selection_one"
72              android:windowSoftInputMode="stateHidden" >
73          </activity>
74          <activity
75              android:name="com.example.sensor.ListSelectionTwo"
76              android:label="@string/title_activity_list_selection_two"
77              android:windowSoftInputMode="stateHidden" >
78          </activity>
79      </application>
80
81  </manifest>
```