

UNIVERZITA PARDUBICE

Fakulta elektrotechniky a informatiky

Mobilní zpracování dat a jejich vyhodnocení

Bc. Jiří Kratochvíl

Diplomová práce

2015

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2014/2015

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Jiří Kratochvíl**
Osobní číslo: **I13418**
Studijní program: **N2646 Informační technologie**
Studijní obor: **Informační technologie**
Název tématu: **Mobilní zpracování dat a jejich vyhodnocení**
Zadávající katedra: **Katedra softwarových technologií**

Z á s a d y p r o v y p r a c o v á n í :

Cílem práce je vytvoření systému aplikací pro zpracování dat z mobilního zařízení na serveru. Teoretická část práce je zaměřena na výběr, popis a porovnání jednotlivých technologií vhodných pro tvorbu podobného systému. Implementovaná serverová část bude získané informace ukládat a zpracovávat. Také umožní uživateli ve webovém prohlížeči zobrazovat a operovat s daty i z více různých mobilních zařízení. Operačním systémem na klientském mobilním zařízení, které je poskytovatelem dat, je zvolen OS Android.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

MEIER, Reto. Professional Android 4 application development. Updated for Android 4. Indianapolis: John Wiley, 2012, 817 p. ISBN 978-111-8262-153.
HARDY, Brian. Android programming: the big nerd ranch guide. 1st ed. Atlanta, GA: Big Nerd Ranch, 2013, 580 p. ISBN 03-218-0433-3.
DEBU PANDA, Reza Rahman. EJB 3 in action. Second edition. Shelter Island: Manning, 2014, 560 s. ISBN 978-193-5182-993.
SATERNOS, Casimir. Client-server Web Apps With Javascript and Java. 1st edition. Sebastopol, CA: O'Reilly Media, 2014, 260 p. ISBN 978-144-9369-330.

Vedoucí diplomové práce:

Ing. Zdeněk Šilar, Ph.D.

Katedra informačních technologií

Datum zadání diplomové práce:

31. října 2014

Termín odevzdání diplomové práce:

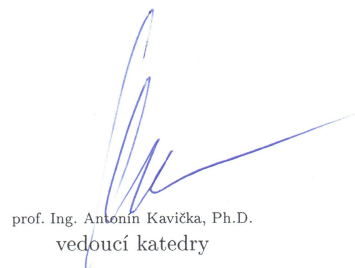
15. května 2015



prof. Ing. Simeon Karamazov, Dr.
děkan



L.S.



prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2014

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 12. 5. 2015

Bc. Jiří Kratochvíl

Poděkování

Chtěl bych poděkovat všem lidem, kteří mě po dobu mého studia podporovali a pomáhali mi překonat veškeré těžkosti. Především bych pak chtěl poděkovat mé přítelkyni za její nesmírnou laskavost a obětavost a mé rodině za vše, co pro mě udělali. Také bych chtěl poděkovat vedoucímu mé práce, Ing. Zdeňku Šilarovi, Ph.D., za jeho ochotu a cenné rady a Čendovi za poskytnutí serveru.

Anotace

Práce se zabývá vytvořením systému aplikací, jenž má za úkol měření veličin na mobilním zařízení, přenos naměřených dat na server a následné zpracování na serveru. Teoretická část obsahuje návrh systému a představuje technologie, které jsou v daných aplikacích použity. V praktické části jsou pak popsány detaily implementací jednotlivých vytvořených aplikací.

Klíčová slova

aplikace, mobilní zařízení, měření, Android, Java

Title

Mobile data processing and evaluation

Annotation

Thesis deals with creation of application system for measurement of quantities on mobile device, transmission of measured data to server and consecutive processing on server. Theoretical part contains system design and presents technologies which are used in aforementioned applications. Practical part describes details of each application's implementation.

Keywords

application, mobile device, measurement, Android, Java

Obsah

Seznam zkratk	8
Seznam obrázků	9
Seznam tabulek	9
Úvod	10
1 Architektura systému	12
1.1 Komponenty systému	12
1.1.1 Senzor	13
1.1.2 Měřicí zařízení	14
1.1.3 Komunikační zařízení	14
1.1.4 Mobilní zařízení	14
1.1.5 Server	14
1.2 Aplikace	14
1.2.1 Mobilní aplikace	15
1.2.2 Serverová komunikační aplikace	15
1.2.3 Webová aplikace	15
2 Použité technologie	16
2.1 Společné technologie	16
2.1.1 Programovací jazyk Java	16
2.1.2 Java Logging API	16
2.1.3 JavaScript, HTML a knihovna ComponentJS	17
2.1.4 Databáze PostgreSQL	18
2.1.5 JPA – EclipseLink	18
2.1.6 Operační systém Debian 7	19
2.2 Mobilní aplikace	19
2.2.1 Operační systém Android	19
2.2.2 Databáze SQLite	20
2.2.3 JPA – OrmLite	20
2.2.4 Knihovna grafů AndroidPlot	20
2.2.5 Programovací jazyk Lua (LuaJ)	20
2.3 Serverová komunikační aplikace	22
2.3.1 NIO	22
2.4 Webová aplikace	22
2.4.1 Aplikační server WildFly 8	22
2.4.2 Webový framework Vaadin	22
2.4.3 Aplikační framework Spring	23
2.4.4 JTA	23
2.4.5 Knihovna grafů Highcharts	23
3 Návrh implementace	25
3.1 Protokol komunikace mobilního a komunikačního zařízení	25
3.2 Protokol komunikace mobilního zařízení a serveru	28

3.3	Databáze mobilního zařízení	31
3.3.1	Tabulka device_messages	32
3.3.2	Tabulka measurements	32
3.3.3	Tabulka measured_values	33
3.3.4	Pohled measurement_values	33
3.3.5	Tabulka server_messages	34
3.4	Databáze serveru	34
3.4.1	Tabulka measurement.measurements	35
3.4.2	Tabulka measurement.measured_values	35
3.4.3	Pohled measurement.measurement_values	36
3.4.4	Tabulka device_configurations.androids	36
3.4.5	Tabulka device_configurations.devices	37
3.4.6	Tabulka device_configurations.sensors	37
3.4.7	Tabulka device_configurations.limits	38
4	Mobilní aplikace	39
4.1	Návrh a implementace mobilní aplikace	39
4.1.1	Služba DeviceCommunicationService	40
4.1.2	Služba ServerCommunicationService	41
4.2	Předvedení mobilní aplikace	44
5	Serverová aplikace	49
5.1	Návrh a implementace serverové aplikace	49
5.1.1	Vstupně/výstupní část	49
5.1.2	Zpracování zpráv	51
5.2	Předvedení serverové aplikace	54
6	Webová aplikace	56
6.1	Návrh a implementace webové aplikace	57
6.1.1	Zobrazení naměřených hodnot — DefaultView	57
6.1.2	Zobrazení konfigurací — AndroidsView	59
6.2	Předvedení webové aplikace	60
	Závěr	64
	Literatura	65
	Příloha A Use case diagram systému aplikací	68
	Příloha B sensorView.js controller	69
	Příloha C sensorView.js model	70
	Příloha D sensorView.js view	71
	Příloha E Model databáze měřicího zařízení	72
	Příloha F Model databáze serveru	73
	Příloha G Kód JavaScriptové komponenty webové aplikace	74

Seznam zkratek

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

CPU Central processing unit

EJB Enterprise JavaBeans

GUI Graphical user interface

IPC Inter-process communication

JAXB Java Architecture for XML Binding

JPA Java Persistence API

JSON JavaScript Object Notation

JTA Java Transaction API

MVC Model-view-controller

NIO New Input/Output (případně Non-blocking Input/Output)

ORM Object-relational mapping

OS Operating system

SPA Single-page application

UI User interface

XML Extensible Markup Language

Seznam obrázků

1	Architektura systému.	13
2	Debugger knihovny ComponentJS.	18
3	Poměr operačních systémů na mobilním trhu.	19
4	Ukázka grafu z knihovny Highcharts.	24
5	Sekvenční diagram odesílání zprávy na komunikační zařízení.	25
6	Binární formát zpráv mobilní zařízení — komunikační zařízení.	27
7	Zpráva z mobilního zařízení pro načtení naměřených dat.	28
8	Zpráva z komunikačního zařízení s naměřenými daty.	28
9	Binární formát zpráv mobilní zařízení — komunikační zařízení.	30
10	Navigace v GUI mobilní aplikace.	44
11	Hlavní obrazovka mobilní aplikace.	45
12	Zobrazení grafu v mobilní aplikaci.	45
13	Obrazovka nastavení mobilní aplikace.	46
14	Obrazovka nastavení komunikace se serverem v mobilní aplikaci.	46
15	Obrazovka s komunikací mobilního a komunikačního zařízení.	47
16	Zobrazení konfiguračního souboru externím textovým editorem.	48
17	Obrazovka webové aplikace s aktuálními hodnotami z mobilních zařízení.	60
18	Obrazovka webové aplikace s grafem všech naměřených hodnot.	61
19	Obrazovka webové aplikace s grafem naměřených hodnot ze senzoru.	62
20	Obrazovka webové aplikace s konfiguracemi mobilních zařízení.	63
21	Obrazovka webové aplikace s konfiguracemi měřicích zařízení.	63

Seznam tabulek

1	Význam polí ve zprávách mobilní zařízení — komunikační zařízení.	27
2	Význam polí ve zprávách mobilní zařízení — server.	30
3	Výčet parametrů serverové aplikace.	54

Úvod

Tato diplomová práce se zabývá tvorbou systému aplikací pro měření hodnot (fyzikálních veličin, stavů sledovaných objektů atd.). Takovéto systémy jsou velice často používány v různých technických odvětvích a nezřídka vyžadují použití specializovaného hardwaru jak pro samotné měření, tak i následné zpracování. Jedním z cílů diplomové práce je nahrazení takového hardwaru mobilním zařízením s operačním systémem Android. Pokud je nutné naměřená data zobrazovat přímo u zdroje měření, je potřeba zařízení s displejem. Případně pokud se mají data odesílat po Wi-Fi, Ethernetu nebo mobilní síti, je opět potřeba hardware, který toto umožní. Vzhledem k současnému rozšíření mobilních zařízení s operačním systémem Android se tato zařízení jeví být vhodným kandidátem. Obsahují jak prostředky pro připojení k přenosovým sítím, tak displej. V podstatě pouze zbývá dodat vhodný software.

První část diplomové práce je teoretická. Zabývá se představením systému pro měření hodnot a technologiemi v tomto systému použitými. První kapitola popisuje architekturu systému z hlediska jednotlivých fyzických komponent a dále z hlediska jednotlivých aplikací, které jsou vytvořeny v praktické části. Druhá kapitola představuje čtenáři technologie použité k tvorbě jednotlivých aplikací.

Druhá část práce je praktická. Obsahuje popis implementačních detailů systému a také představení jednotlivých vytvořených aplikací. V třetí kapitole práce jsou popsány detaily systému aplikací z hlediska implementace. Zabývá se zejména protokoly pro komunikaci mezi aplikacemi a návrhem databází. Čtvrtá kapitola obsahuje návrh, implementaci a představení mobilní aplikace. Pátá kapitola detailněji popisuje serverovou aplikaci pro komunikaci serveru s mobilními zařízeními. Poslední, šestá kapitola prezentuje návrh, implementaci a předvedení webové aplikace pro práci s naměřenými údaji.

Typografické konvence

V diplomové práci jsou použity následující typografické konvence:

promenna Neproporcionální písmo je využito pro názvy proměnných.

ClassName Kurzívou jsou psány názvy metod, typů a tabulek.

Ukázky kódu v textu jsou psány neproporcionálním písmem v rámečku a se zvýrazněnou syntaxí.

```
public class Ukazka {  
  
    public static void main(String[] args) {  
        :  
    }  
  
}
```

Ukázky příkazů spuštěných v příkazovém řádku jsou v rámečku neproporcionálním písmem. Prompt příkazového řádku je značen symbolem \$. Mezi samotným příkazem a výstupem programu je pro přehlednost vynechán řádek.

```
$ uname -a
```

```
Linux Taliesin 4.0.1-1-ARCH #1 SMP PREEMPT Wed Apr 29 12:00:26  
CEST 2015 x86_64 GNU/Linux
```

1 Architektura systému

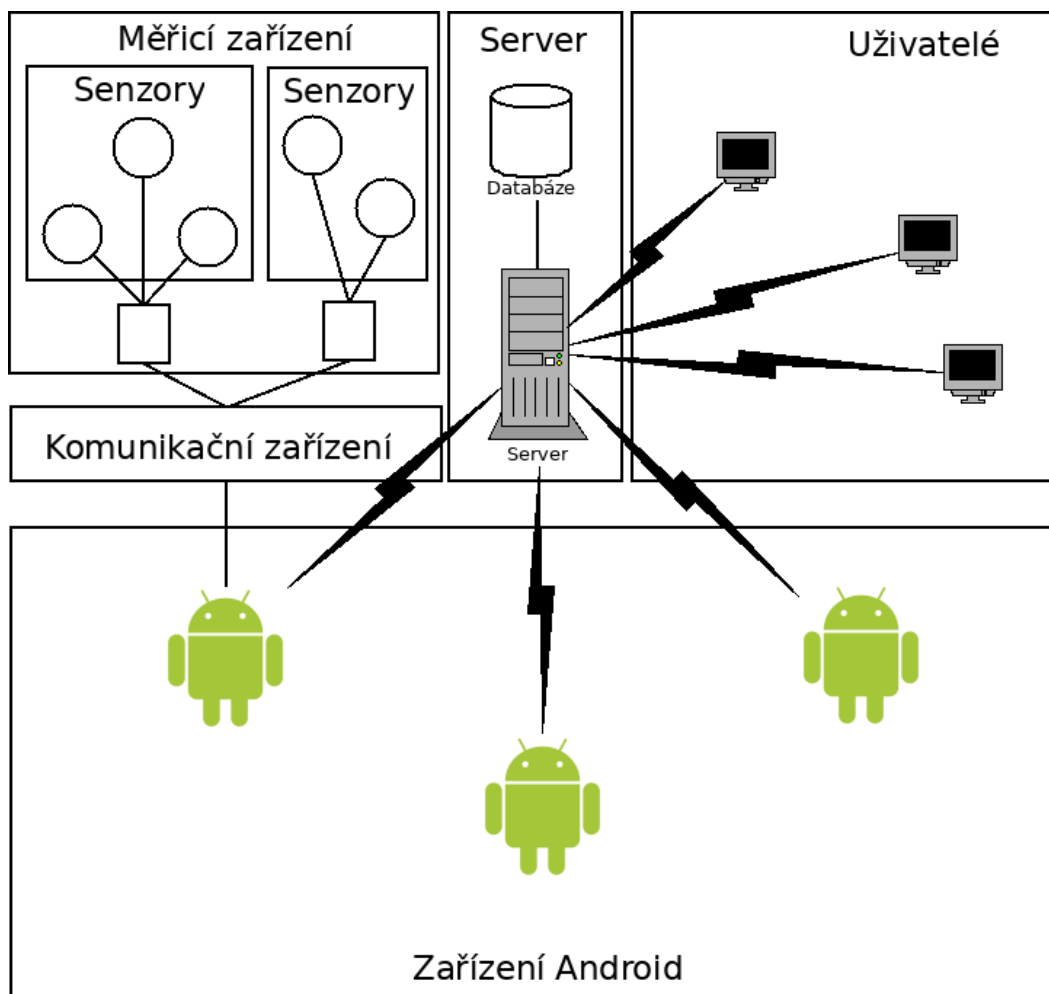
V této kapitole bude popsána architektura celého systému měření. První část se zabývá základními stavebními prvky a bude v ní představeno uspořádání měřicích přístrojů a propojení jednotlivých komponent systému od senzorů až po server. Druhá část obsahuje základní popis vytvořeného softwaru, který je v systému použit.

1.1 Komponenty systému

Název této podkapitoly mohl být „Hardware“, avšak toto označení by bylo mírně zavádějící. Jednotlivé měřicí komponenty nemusí být reprezentovány pouze hardwarovými zařízeními, ale mohou být i čistě softwarové, jak bude možné vidět v následujících kapitolách. Jedná se tedy spíše o měřicí zařízení z abstraktního hlediska, konkrétní implementace může být různá.

Za účelem pochopení toho, jak pracují aplikace vytvořené v této práci, je nejprve třeba seznámit čtenáře se základními stavebními kameny celého systému měření. Popis jednotlivých komponent bude postupovat od těch, které jsou nejbližší sledovaným entitám až po ty nejvzdálenější, které jsou tedy blíže k uživateli pracujícím s webovým prohlížečem.

Nejprve však bude předveden celkový pohled na architekturu systému, jelikož při seznamování čtenáře s jednotlivými komponentami je třeba, aby měl představu o tom, na jakém místě v systému se nacházejí. Všechny komponenty mají hierarchické stromové uspořádání, kde senzory náleží k měřicímu zařízení, měřicí zařízení náleží k mobilním zařízením a ty zase k serveru, na který odesílají data. Toto je znázorněno na následujícím obrázku 1.



Obrázek 1 – Architektura systému.

1.1.1 Senzor

Hierarchicky nejnižší položenou komponentou v systému je senzor. Jedná se o prvek, který zaznamenává požadovanou veličinu. Každý senzor má přiřazeno číslo v rozsahu 0–32767 (v Javě je reprezentováno 16 bitovým znaménkovým číslem). Pro zjednodušení je výstup senzorů typu float, tedy 32 bitové číslo s plovoucí řádovou čárkou. Tento datový typ by měl být postačující pro zaznamenání jednoduchých naměřených hodnot velkého množství různých typů senzorů. Jelikož s naměřenými hodnotami nejsou prováděny žádné aritmetické operace, přesnost tohoto typu postačuje. V diplomové práci jsou pro ukázkou použity následující typy senzorů:

- teplotní (Temperature),
- digitální (Digital).

1.1.2 Měřicí zařízení

Měřicí zařízení je abstrakce, která seskupuje více senzorů do jednoho celku, jenž může reprezentovat složenou hodnotu měřené entity. Může se například jednat o měřicí zařízení, které zároveň reprezentuje tělesnou teplotu, krevní tlak a tepovou frekvenci jednoho člověka. Tento celek je potom označen identifikačním číslem v rozsahu 0–32767 (v Javě 16 bitovým znaménkovým číslem).

1.1.3 Komunikační zařízení

Komunikační zařízení představuje prvek, který seskupuje měřicí zařízení a zároveň představuje rozhraní pro komunikaci s mobilním zařízením. Tento prvek může být reprezentovaný přímo v mobilní aplikaci nebo se může jednat o hardwarové zařízení. Hlavní funkcí je zprostředkování interakce mezi mobilní aplikací a měřicími zařízeními. Aplikace posílá zprávy komunikačnímu zařízení a to odpovídá příslušným způsobem. Například pokud chce získat mobilní aplikace nové hodnoty měření, pošle zprávu s dotazem na nové hodnoty komunikačnímu zařízení. To získá aktuální hodnoty ze senzorů a ve zprávě v příslušném formátu pošle odpověď aplikaci.

Jednotlivé senzory jsou reprezentovány v rámci komunikačního zařízení formou unikátního identifikátoru. Tím je 32 bitové číslo, které je získáno kombinací identifikátoru senzoru a identifikátoru zařízení. Více informací bude uvedeno v kapitolách o komunikačním protokolu a databázích.

1.1.4 Mobilní zařízení

Mobilním zařízením je myšlen tablet nebo mobilní telefon s operačním systémem Android. Toto zařízení je spojeno s komunikačním zařízením, ze kterého získává údaje o příslušných senzorech. Tyto informace jsou lokálně uchovávány v mobilním zařízení a také odesílány na server k dalšímu zpracování.

1.1.5 Server

Server je počítač, na kterém běží služby pro komunikaci s mobilními zařízeními, ukládání přijatých dat a prezentaci dat uživatelům skrze webovou aplikaci. Systém aplikací pro měření může být rozprostřen i na více serverů, jelikož se skládá z více samostatných programů. Více bude uvedeno v následující kapitole.

1.2 Aplikace

V této kapitole budou popsány jednotlivé aplikace, ze kterých se celý systém měření skládá. Budou popsány opět od těch nejbližších k jednotlivým senzorům až k těm nejdál, tedy nejbližším k uživateli. Celkem jsou tři hlavní aplikace — pro mobilní zařízení, pro

komunikaci serveru a mobilního zařízení a pro webové stránky (aplikace ve webovém prohlížeči). Use case diagram celého systému těchto aplikací je k dispozici v příloze A.

1.2.1 Mobilní aplikace

Mobilní aplikace běží na zařízeních se systémem Android. Jejím hlavním úkolem je získávat informace z komunikačního zařízení, ukládat je do interní databáze a odesílat na server. Těmito informacemi jsou myšleny jak aktuální hodnoty senzorů, tak i typy senzorů, jejich příslušnost k daným měřicím zařízením a další informace týkající se konfigurace. Navíc aplikace umožňuje zobrazení aktuálních a historických hodnot uživateli.

1.2.2 Serverová komunikační aplikace

Serverová komunikační aplikace, jak už název napovídá, má za úkol komunikovat. Konkrétně komunikuje s připojenými mobilními zařízeními. Stará se o zpracování příchozích zpráv a ukládání informací z nich do databáze. Tato databáze může být klidně na jiném serveru, stačí specifikovat cílovou adresu a port databáze, ke které se má program připojovat.

1.2.3 Webová aplikace

Webová aplikace slouží ke zobrazení naměřených dat a dalších informací koncovému uživateli, pracujícímu s webovým prohlížečem. Výhodou takovéto aplikace je, že ji stačí mít pouze na jednom serveru, ke kterému se uživatelé připojují z prohlížeče. Tím odpadá starost o více instancí aplikace na různých počítačích a problémy s přenositelností na různých platformách. Je však pravda, že Java řeší právě problém přenositelnosti i u desktopových aplikací. Avšak stále by byl problém s údržbou, aktualizací a podobnými činnostmi. Takto stačí mít na každém počítači běžný webový prohlížeč, který je již stejně v základu téměř každého operačního systému nainstalován.

2 Použité technologie

Tato kapitola se zabývá představením technologií použitých v diplomové práci. Nejprve budou představeny ty, které jsou společné pro více programů. Potom budou v příslušných sekcích uvedeny ty, které jsou unikátní pro danou aplikaci. Společným znakem pro všechny technologie je to, že se jedná o svobodný software, jehož kód je tudíž otevřený.

2.1 Společné technologie

Přehled společných technologií, které umožňují základní funkčnost aplikací.

2.1.1 Programovací jazyk Java

Ve všech programech je převážně používán programovací jazyk Java, který je zastoupen ve vícero verzích. V mobilní aplikaci je z důvodu kompatibility se staršími verzemi systému Android (zpětně až k verzi 2.3 Gingerbread) použita verze 6. V aplikaci na server, která komunikuje se zařízeními Android, je použita verze 7. Ve webové aplikaci je použita verze 8 z důvodu častého využití ad-hoc listenerů, které lze mnohdy lambda výrazy reprezentovat přehledněji než anonymními třídami. To je znázorněno v následující ukázce.

```
private void handleEvent(ValueChangeEvent event) {
    // Zpracování objektu události.
}

// Listener ve formě lambda výrazu.
table.addValueChangeListener(
    event -> handleEvent(event)
);

// Listener ve formě anonymní třídy.
table.addValueChangeListener(new ValueChangeListener() {
    @Override
    public void valueChange(ValueChangeEvent event) {
        handleEvent(event);
    }
});
```

2.1.2 Java Logging API

Pro záznam do logu bylo použito Java Logging API. To je obsaženo v každém prostředí Java a umožňuje výstup ladících informací například do konzole nebo do

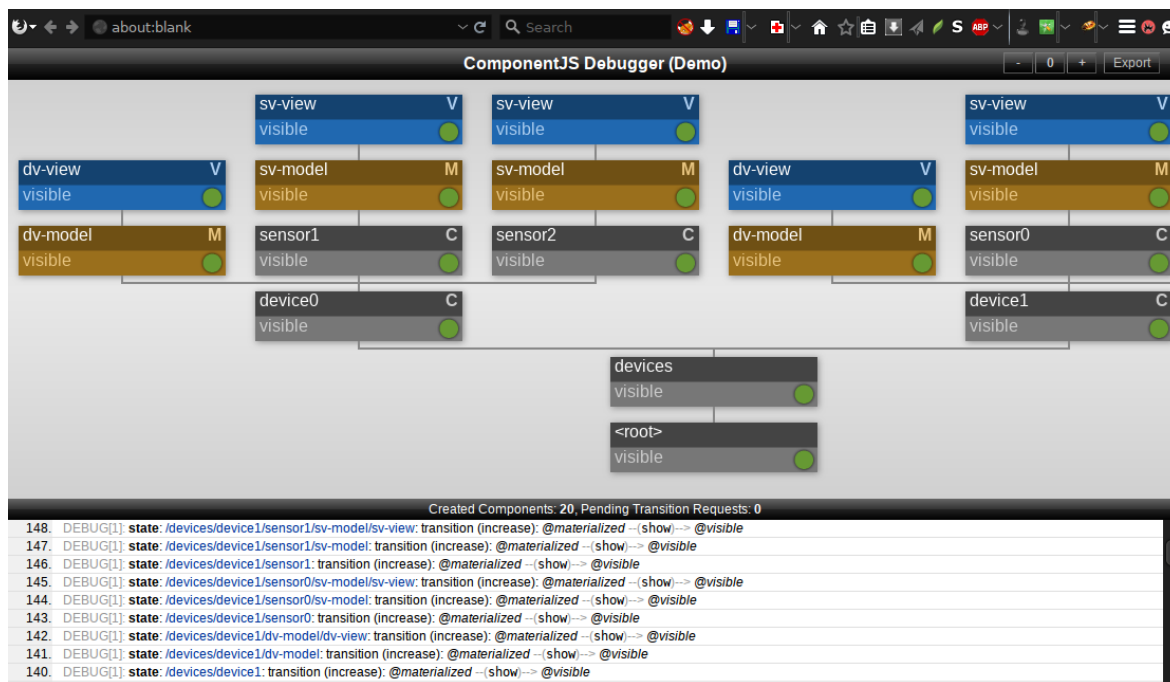
souboru. Skrze toto API lze zaznamenávat různé události v rámci aplikace, o kterých by měl být záznam — například různá varování, chyby a tak dále. Ke každému záznamu je přiřazena úroveň — jestli se jedná právě o informativní záznam nebo například záznam chyby. Tyto úrovně jsou seřazeny podle důležitosti informace, kterou reprezentují, což umožňuje řídit od jaké úrovně výše se budou záznamy zobrazovat nebo ukládat. K dispozici jsou i převodníky pro formát záznamů a další užitečné konstrukce pro práci s logy.

2.1.3 JavaScript, HTML a knihovna ComponentJS

HTML a JavaScript jsou samozřejmě ve velké míře používány ve webové aplikaci. Avšak v kombinaci s JavaScriptovou knihovnou ComponentJS byly tyto technologie uplatněny i pro část grafického rozhraní mobilní aplikace. To umožňuje použití víceméně stejného kódu v obou aplikacích a vede k jednotnému vzhledu grafického rozhraní, což je žádoucí jak z uživatelského hlediska, kdy se uživatel nemusí učit orientovat v různých rozhraních, ale i pro vývojáře, jelikož změna na jednom místě kódu ovlivní více aplikací najednou.

Jelikož má mobilní i webová aplikace vlastní HTML stránku pro zobrazení aktuálních hodnot senzorů, tak stále zůstává možnost provádět úpravy cílené na jednotlivé aplikace. Stačí připojení různých CSS souborů v různých aplikacích. Knihovna ComponentJS zprostředkovává run-time komponentový systém pro hierarchicky strukturovaná grafická prostředí, zejména tzv. SPA aplikace, tedy takové, jejichž celé UI je na jedné stránce. Při reprezentaci jednotlivých komponent je využita MVC architektura se sofistikovanými prostředky například pro reprezentaci událostí, služeb a dalších mechanismů (*ComponentJS* 2009—2014). Ukázku, jak vypadá reprezentace senzoru, můžeme vidět v přílohách B, C a D, které popořadě odpovídají komponentám pro controller, model a view.

Tato knihovna má také zabudovaný debugger, který dokáže celou hierarchii komponent přehledně zobrazit, jak můžeme vidět na obrázku 2.



Obrázek 2 – Debugger knihovny ComponentJS.

2.1.4 Databáze PostgreSQL

Jako hlavní databáze byla zvolena PostgreSQL zejména kvůli jazyku PL/pgSQL, který je podobný procedurálnímu jazyku PL/SQL firmy Oracle (THE POSTGRESQL GLOBAL DEVELOPMENT GROUP, 1996—2015b). Jedná se o objektově relační databázi, jejíž první verze byla vydána v roce 1996 (THE POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2008) a která vznikla z projektu Ingres v Kalifornské univerzitě v Berkeley. Je distribuována pod licencí PostgreSQL License, což je open source licence, podobná jako BSD nebo MIT licence (THE POSTGRESQL GLOBAL DEVELOPMENT GROUP, 1996—2015a).

Pro práci s touto databází byl použit open source nástroj pgModeler, ve kterém lze modelovat databázi a generovat SQL kód.

2.1.5 JPA – EclipseLink

Pro perzistenci objektů do databáze byla využita technologie JPA, konkrétně verze 2.1. Za tímto účelem byl použit projekt EclipseLink, který je implementací JPA a JAXB a pro JPA 2.0 je dokonce implementací referenční (THE ECLIPSE FOUNDATION, 2008). Je postaven na projektu TopLink od firmy Oracle (ORACLE, 2007). Tuto knihovnu používá jak serverová aplikace pro komunikaci s mobilními zařízeními, tak webová aplikace. Obě dvě ji využívají pro práci s konfigurací mobilního zařízení.

2.1.6 Operační systém Debian 7

Serverové aplikace byly nasazeny na server s operačním systémem Debian 7. Jedná se o jednu z neznámějších distribucí Linuxu, jejíž počátky sahají až do roku 1993 (DEBIAN DOCUMENTATION TEAM, 2013) a která patří mezi nejpoužívanější distribuce vůbec, ať už jako OS na serveru či desktopech (W3TECHS, 2009—2015).

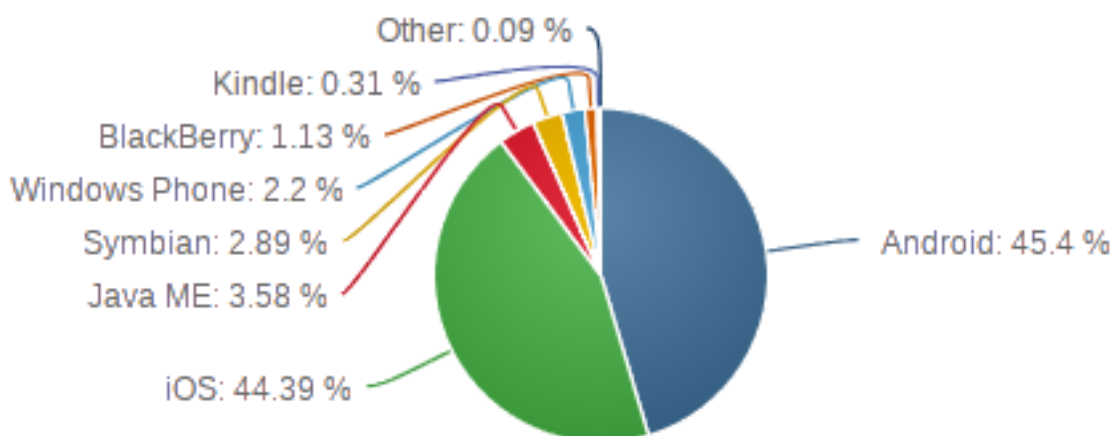
2.2 Mobilní aplikace

Tato kapitola se zabývá představením technologií použitých v aplikaci pro mobilní zařízení.

2.2.1 Operační systém Android

Základní softwarovou výbavou mobilního zařízení je operační systém Android, který je v současné době jednou z nejrozšířenějších mobilních platform (NET APPLICATIONS.COM, 2006—2015). Je postaven na OS Linux a programovacím jazyku Java. Byl zvolen jako mobilní platforma této práce, jelikož je nejsnáze dostupným systémem. Lze se s ním setkat na velkém množství mobilních zařízení — od těch nejlevnějších až po velmi drahé. Zatímco na světovém trhu je poměr Android versus iOS celkem vyrovnaný, na českém trhu jasně převládá Android (NET APPLICATIONS.COM, 2006—2015; GEMIUS SA, 2000—2015).

Poměr operačních systémů na mobilním trhu mezi dubnem 2014 a dubnem 2015 lze vidět na následujícím obrázku 3. Tento graf pochází ze zdroje (NET APPLICATIONS.COM, 2006—2015).



Obrázek 3 – Poměr operačních systémů na mobilním trhu.

Alternativní platformou, kterou by bylo možné zvolit v této práci je iOS. Ve světě i České Republice se jedná o druhý nejrozšířenější mobilní operační systém, ačkoliv rozdíl mezi podílem na mobilním trhu v ČR a ve světě je markantní. Za systémem iOS

stoi společnost Apple Inc. Její produkty jsou známé pro svou pokrokovost a dobré zpracování jak z hlediska hardwaru, tak softwaru. Neméně proslulé jsou však i svou cenou, což z nich nedělá vhodnou platformu pro tuto práci.

Dalším kandidátem je platforma Windows od firmy Microsoft. V současné době je podíl tohoto operačního systému na mobilním trhu nepatrný, avšak na desktopech se jedná o nejrozšířenější operační systém v ČR (GEMIUS SA, 2000—2015). Ve verzi Windows 10 se však Microsoft snaží sjednotit operační systém pro desktop a pro mobilní zařízení (DUDLEY, 2014). Výsledkem by měla být univerzální aplikační architektura pro všechna zařízení s Windows, o což už se Microsoft snaží již ve verzi Windows 8.1 a Windows Phone 8.1 (WIGLEY, 2014). Stále se však jedná o dva rozdílné operační systémy. Pokud se záměr jednoho operačního systému vydaří, mohl by se tento systém stát velkou konkurencí výše zmíněným mobilním systémům.

2.2.2 Databáze SQLite

Pro uložení dat v mobilní aplikaci byla použita databáze SQLite, která je integrovaná v OS Android. Jedná se tedy o tzv. embedded databázi. Její největší předností je velikost, která nabývá hodnot přibližně od 340 KiB až do 4,7 MiB v závislosti na použité platformě (*SQLite Download Page* 2015). V databázovém systému je implementována většina z SQL standardu.

2.2.3 JPA – OrmLite

Stejně jako na serveru, i na mobilním zařízení je použita technologie JPA pro uložení konfigurace. Zde je však zvolena knihovna OrmLite, která pracuje nad databází SQLite a která má také velmi nízké paměťové nároky, což je na mobilním zařízení prioritou. Velikost všech potřebných souborů nepřesahuje 400 KiB (WATSON, 2015).

2.2.4 Knihovna grafů AndroidPlot

K vykreslení naměřených hodnot na mobilním zařízení byla použita knihovna AndroidPlot. S její pomocí lze vytvářet množství různých grafů od liniových až ke koláčovým.

2.2.5 Programovací jazyk Lua (LuaJ)

Zajímavou technologií použitou v mobilní aplikaci je multiparadigmatický dynamicky typovaný programovací jazyk Lua, v tomto případě reprezentovaný knihovnou LuaJ. Jedná se o velmi odlehčený jazyk, jehož interpret je napsán v ANSI C. Lze jej tedy bez problémů zkompileovat na jakékoli platformě se standardním C kompilátorem. Velkou výhodou je také malá velikost interpretu, jehož zdrojový kód i s dokumentací a binárními soubory pro jisté platformy se vejde i na disketu (IERUSALIMSKY, 2013), což z něj dělá ideálního kandidáta pro embedded programovací jazyk.

V programu pro mobilní zařízení je tento jazyk použit pro popis konfigurace připojeného zařízení. Jelikož není připojováno žádné reálné hardwarové zařízení, emuluje jeho funkci interpret jazyka Lua, který je zastoupen knihovnou LuaJ. Tato knihovna má, stejně tak jako původní interpret, minimální velikost. Jak se můžeme přesvědčit na úložišti Maven, velikost knihovny je 331 KiB bez jakýchkoliv závislostí na jiných knihovnách (MVNREPOSITORY, 2014).

Přes svou malou velikost, velmi jednoduchou syntaxi a omezený počet základních datových struktur (Lua obsahuje pouze jedinou datovou strukturu, kterou je tabulka) se jedná o velmi mocný programovací jazyk. Jsou v něm obsaženy základní kameny pro objektově orientované a funkcionální programování, avšak je již na uživateli, aby implementoval další potřebné věci. Například pro reprezentaci objektů se užívají tabulky. V jazyku Lua jsou všechny funkce tzv. first-class funkce, což znamená, že je lze přiřazovat do proměnných, předávat jako argumenty, vracet z funkcí a obecně s nimi provádět stejné operace jako s běžným datovým typem. Toho lze využít k uložení funkcí do tabulky, kde poté reprezentují metody objektu.

Jak již bylo zmíněno, Lua je použita pro popis konfigurace zařízení. Jelikož je v tom případě konfigurační soubor souborem zdrojovým, existuje jistá výhoda oproti „statické“ konfiguraci například ve formátu XML nebo JSON. V konfiguračním souboru můžeme přímo napsat funkci, která bude generovat naměřené hodnoty v závislosti na čase. Takovéto využití můžeme vidět v následující ukázce.

```
Conf = require "configuration"
local sensorsDevice1 = {
    {
        id = 1,
        valueType = Conf.valueType.TEMPERATURE,
        generator = function(time)
            local perioda = 10 -- Perioda v minutách.
            time = time / 1000 / 60 -- Čas z ms na min.
            local x = ((2 * math.pi) / perioda) * (time %
                perioda)
            -- Kmit +-20, posun +10.
            return math.sin(x) * 20 + 10
        end
    },
    {
        id = 2,
        valueType = Conf.valueType.TEMPERATURE,
        generator = function(time)
            local perioda = 25 -- Perioda v minutách.
            time = time / 1000 / 60 -- Čas z ms na min.
            local x = ((2 * math.pi) / perioda) * (time %
                perioda)
            -- Kmit +-5, posun +18.
            return math.sin(x) * 5 + 18
        end
    }
}
```

Proměnná `sensorsDevice1` reprezentuje jedno zařízení, ve kterém jsou obsaženy dva senzory, jejichž popis je uvnitř složených závorek. U prvního senzoru můžeme vidět, že k „proměnné“ `id` je přiřazena hodnota 1, k `valueType` pro změnu „konstanta“ značící, že typ hodnoty je teplota. U „proměnné“ `generator` můžeme vidět přiřazenou funkci, která generuje hodnoty teploty v závislosti na čase, zadaném jako vstupní parametr. Výstupem budou hodnoty, které budou v průběhu času tvořit sinusoidu.

2.3 Serverová komunikační aplikace

V této kapitole budou představeny technologie, které jsou použity v serverové aplikaci komunikující se zařízeními Android.

2.3.1 NIO

Nejdůležitější součástí serverové komunikační aplikace je technologie NIO. Jedná se o soubor API pro škálovatelné vstupně/výstupní operace, rychlé vstupně/výstupní operace s binárními a znakovými daty s použitím mezipaměti, regulární výrazy, konverzi znakových sad a vylepšené rozhraní souborového systému (ORACLE CORPORATION AND/OR ITS AFFILIATES, 2014).

Z výše zmíněných funkcionalit jsou používány zejména neblokující vstupně/výstupní operace za využití takzvaných kanálů (Channels), které pracují na nižší úrovni než standardní knihovny pro vstup/výstup.

2.4 Webová aplikace

Poslední část je věnována technologiím, které jsou využity ve webové aplikaci.

2.4.1 Aplikační server WildFly 8

Za aplikační server byl zvolen server WildFly 8 od firmy Red Hat především z důvodu vlastních pozitivních zkušeností. Jedná se o server původně vyvíjený firmou JBoss, která byla v roce 2006 zakoupena právě již zmíněnou firmou Red Hat (RED HAT, INC, 2006).

2.4.2 Webový framework Vaadin

Pro tvorbu webové aplikace byl vybrán Java framework Vaadin z důvodů vlastní zkušenosti a také proto, že už v základu zahrnuje mnoho užitečných grafických komponent. Distribuován je pod licencí Apache License 2.0, takže se jedná o open source software. Nevýhodou je však to, že některé zajímavé nástroje jsou dostupné pouze po zakoupení, jako například knihovna s grafy Highcharts, toolkit pro optimalizaci na mobilní zařízení,

nástroj pro testování aplikací a další podobné nástroje, spíše určené pro profesionální vývojáře. Všechny tyto placené přídatky jsou placeny za jednoho vývojáře na jeden měsíc. Celkem tedy mohou vyjít na poměrně velké množství peněz (VAADIN LTD., 2015). Také si lze pořídit placenou podporu. Na druhou stranu jsou tyto placené nástroje spíše „bonusové“ a na oficiálních stránkách lze nalézt repozitář s knihovnami, které jsou převážně vytvořeny přímo samotnými uživateli a jsou k dispozici zdarma.

Tento framework je poměrně výjimečný mezi ostatními podobnými frameworky. Pro tvorbu prezentační vrstvy se nemusí psát XML, XHTML ani HTML stránky, jako je tomu například v případě frameworků JSF, JSP nebo ADF. Veškerá tvorba grafického uživatelského rozhraní aplikace probíhá vytvářením příslušných Java tříd, což dává velkou volnost při tvorbě dynamického GUI.

Webová aplikace se v tomto frameworku skládá ze dvou částí, a to ze serverové a klientské. Většina aplikační logiky běží na serveru, která s klientskou částí komunikuje prostřednictvím technologie Ajax. Výsledné stránky jsou velice dynamické a interaktivní. Pro tvorbu klientské části je využit nástroj Google Web Toolkit, díky kterému je možné vytvářet další vlastní komponenty, pokud by již předpřipravené nestačily. Výhodou je také poměrně snadná integrace již existujících GWT komponent (ENGLUND, 2012).

2.4.3 Aplikační framework Spring

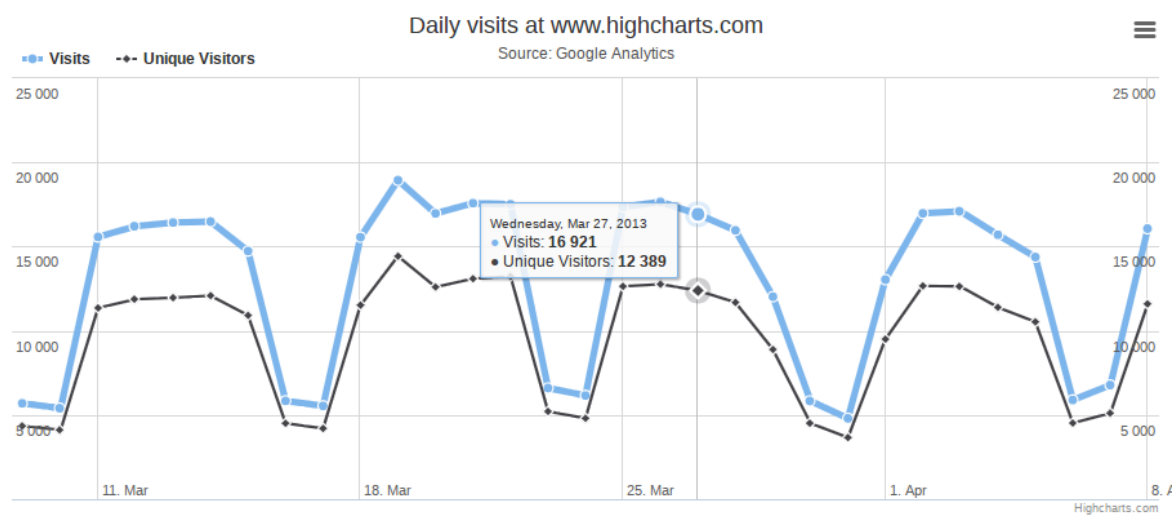
Ve webové aplikaci byl použit framework Spring zejména pro dependency injection a správu transakcí JPA přes JTA. Jedná se o open source aplikační framework a kontejner pro inversion of control na platformě Java, který se dá využít jako alternativa k EJB. Jeho hlavní funkcionalitu lze však využít pro jakoukoliv aplikaci napsanou v jazyce Java, nejen na platformě Java EE. Mezi poskytované služby tohoto frameworku patří například aspektově orientované programování, autentizace a autorizace v aplikaci, správa přístupu k datovým zdrojům, již zmíněný inversion of control kontejner, práce se zprávami, správa transakcí, MVC framework a další (WIKIPEDIA, 2015).

2.4.4 JTA

Pro správu transakcí JPA byla použita technologie JTA, což vedlo k zjednodušení práce s perzistentními daty. Ve frameworku Spring, který je v této práci použitý, je totiž JTA doporučenou technologií namísto ručního řízení transakcí (FERREIRA, 2014).

2.4.5 Knihovna grafů Highcharts

Highcharts je knihovna pro tvorbu grafů napsaná v čistém JavaScriptu. Její první verze byla vydána v roce 2009 (HIGHCHARTS, 2015a) a pro nekomerční využití je zdarma. K dispozici jsou také zdrojové kódy, které je možné upravovat při komerčním i nekomerčním využití. Knihovna podporuje velké množství různých typů grafů a je kompatibilní se všemi aktuálními webovými prohlížeči (HIGHCHARTS, 2015b).



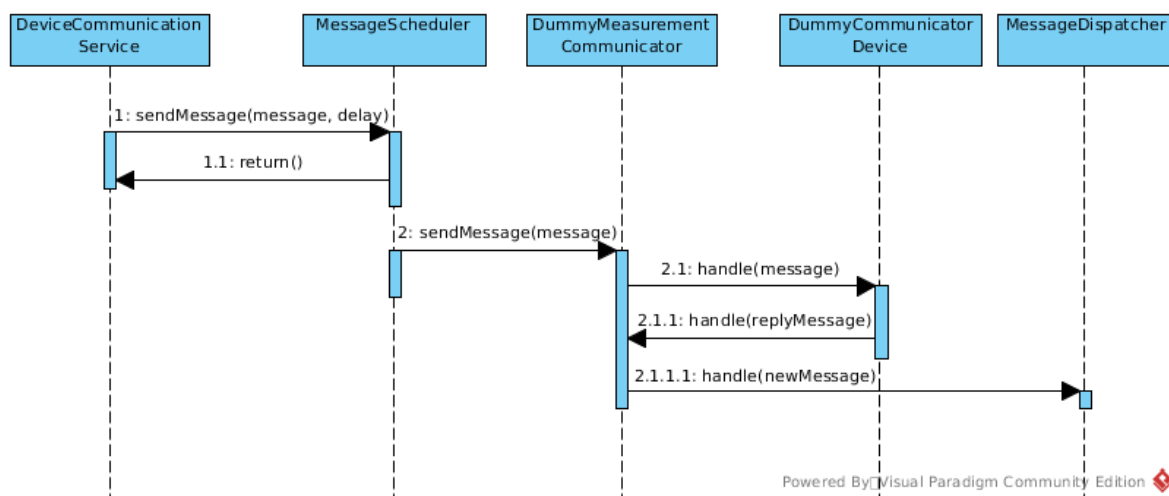
Obrázek 4 – Ukázka grafu z knihovny Highcharts.

3 Návrh implementace

Tato kapitola se zabývá implementačními detaily systému aplikací pro měření. První část pojednává o protokolu komunikace mezi mobilním zařízením a komunikačním zařízením, představuje formu a typy zpráv, které si mezi sebou tyto dvě komponenty měřicího systému posílají. Druhá část se orientuje na popis protokolu komunikace mezi mobilním zařízením a serverem, přesněji serverovou aplikací pro komunikaci s mobilními zařízeními. Budou uvedeny typy zpráv, které se při této komunikaci využívají a jakou formou komunikace probíhá. V následující části bude pojednáno o databázových modelech, které jsou v aplikacích využity. První kapitola o databázích popisuje databázový model použitý na mobilním zařízení. Druhá kapitola o databázích se zabývá databázovým modelem na serveru.

3.1 Protokol komunikace mobilního a komunikačního zařízení

V této kapitole bude popsán protokol komunikace mezi mobilním zařízením a komunikačním zařízením. Jelikož je použito virtuální komunikační zařízení (virtuální jsou i měřicí zařízení a senzory), je protokol komunikace lehce zjednodušen. Celá komunikace probíhá pouze v rámci aplikace a tak tedy zasílání zpráv i jejich přijetí a zpracování je řešeno jedním voláním funkce pro odeslání zprávy. Díky tomu stačí používat pro celou logiku komunikace jedno vlákno. Také byla ze zprávy vypuštěna informace o její délce, protože všechny zprávy jsou už při přijetí do aplikace v podstatě zpracovány v tom ohledu, že je není třeba načítat a parsovat z příchozího proudu bytů, kde není poznat kde jedna zpráva končí a druhá začíná. Každé zprávě už totiž odpovídá jeden objekt v aplikaci, stejně tak, jako by už byly zprávy rozparsovány. Jak komunikace probíhá je názorně ukázáno v sekvenčním diagramu na obrázku 5.



Obrázek 5 – Sekvenční diagram odesílání zprávy na komunikační zařízení.

Více bude o komunikaci mezi mobilním a komunikačním zařízením uvedeno v následujících kapitolách. Nyní bude popsán formát zpráv, které si tyto dvě zařízení vyměňují. V níže uvedené ukázce kódu je k vidění část třídy *MessageMeric*, která obsahuje atributy a kód pro serializaci této třídy do pole bytů. Toto pole je obsah zprávy tak, jak je přenášena mezi zařízeními.

```
public class MessageMeric implements Serializable, Bytes {

    private MessageType messageType;
    private MessageObject messageObject;
    private byte[] data;

    :

    public byte[] toBytes() {
        ByteArrayOutputStream byteOutputStream = new
            ByteArrayOutputStream(256);
        DataOutputStream dos = new
            DataOutputStream(byteOutputStream);
        try {
            dos.writeByte(messageType.getValue());
            dos.writeByte(messageObject.getValue());
            dos.write(data);
            dos.flush();
            return byteOutputStream.toByteArray();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                dos.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        return null;
    }

    :

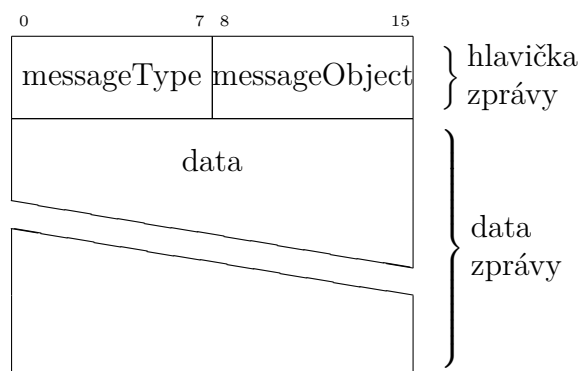
}
```

Popis jednotlivých atributů zprávy, a jakých hodnot mohou nabývat, je uveden v následující tabulce 1.

Binární formát zprávy je uveden na následujícím obrázku 6.

Tabulka 1 – Význam polí ve zprávách mobilní zařízení — komunikační zařízení.

atribut	hodnota	popis
messageType	byte	Typ zprávy.
	QUERY(0)	Zpráva s dotazem z mobilní aplikace na komunikační zařízení.
	REPLY(1)	Zpráva s odpovědí z komunikačního zařízení do mobilní aplikace.
	UNKNOWN(127)	Neznámý typ zprávy, pravděpodobně neimplementovaný.
messageObject	byte	Objekt zprávy - data, která nese.
	DATA(0)	Zpráva s daty, která obsahují naměřené hodnoty z komunikačního zařízení.
	CONFIGURATION(1)	Zpráva s konfigurací komunikačního zařízení.
	UNKNOWN(127)	Neznámý typ zprávy, pravděpodobně neimplementovaný.
data	byte[]	Tělo zprávy — binární data.



Obrázek 6 – Binární formát zpráv mobilní zařízení — komunikační zařízení.

Komunikace probíhá formou dotaz — odpověď. Mobilní zařízení odešle zprávu komunikačnímu zařízení s polem zprávy `messageType` nastaveným na dotaz a polem `messageObject` nastaveným na požadovaná data. Takováto zpráva je ukázána na obrázku 7.

messageType : 8b 0	messageObject : 8b 0	data : 0b null
------------------------------	--------------------------------	--------------------------

Obrázek 7 – Zpráva z mobilního zařízení pro načtení naměřených dat.

Komunikační zařízení poté odpoví zprávou, která má pole `messageType` nastavené na odpověď a `messageObject` ponechané na stejnou hodnotu, jakou měl dotaz. V datové části zprávy jsou potom obsaženy data odpovědi. Na následujícím obrázku 8 je možné vidět odpověď obsahující jedinou hodnotu 15,50. Tato hodnota má velikost 32 bitů, jelikož se jedná o hodnotu javoovského typu `float`.

messageType : 8b 1	messageObject : 8b 0	data : 32b 15.50
------------------------------	--------------------------------	----------------------------

Obrázek 8 – Zpráva z komunikačního zařízení s naměřenými daty.

3.2 Protokol komunikace mobilního zařízení a serveru

Tato kapitola se zabývá popisem komunikace mezi mobilními zařízeními a serverovou komunikační aplikací. Protokol použitý pro komunikaci je velice podobný tomu použitému pro komunikaci mezi mobilním zařízením a komunikačním zařízením. Jelikož se však jedná o komunikaci po síti, je třeba protokol rozšířit o několik dalších polí. V první řadě přibude pole pro délku zprávy. Protože se také zprávy na server posílají v dávkách a tehdy, je-li spojení se serverem dostupné, je u každé zprávy přidán záznam s datem. To potom může být interpretováno libovolně pro různé typy zpráv, většinou však uvádí čas vzniku zprávy na mobilním zařízení. Důležitým polem zprávy je také typ zprávy. Ukázku zdrojového kódu třídy *MessageServer* i se serializací do pole bytů lze vidět níže.

```

public class MessageServer implements Bytes {

    private static final int BASE_LENGTH = (Byte.SIZE +
        Long.SIZE) / 8;
    private int length = BASE_LENGTH;

    private MessageType messageType;
    private long date;
    private byte[] data;

    :

    @Override
    public byte[] toBytes() {
        ByteArrayOutputStream bos = new
            ByteArrayOutputStream(256);
        DataOutputStream dos = new DataOutputStream(bos);
        try {
            dos.writeInt(length);
            dos.writeByte(messageType.getValue());
            dos.writeLong(date);
            if (data != null) {
                dos.write(data);
            }
            dos.flush();
            return bos.toByteArray();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                dos.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        return null;
    }

    :

}

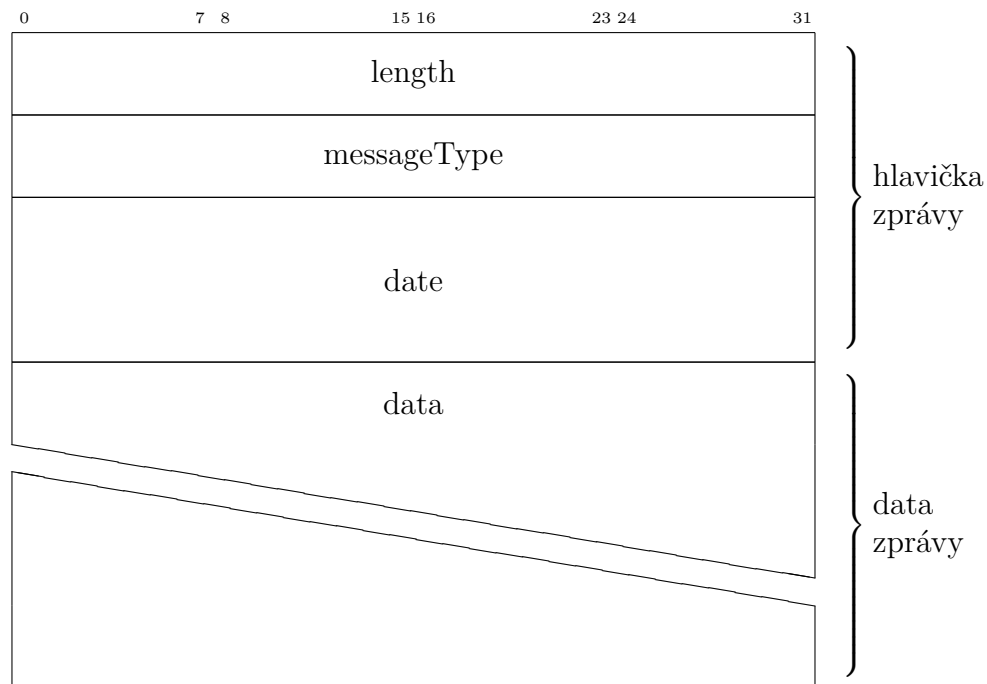
```

Popis polí zprávy a jejich možných hodnot obsahuje tabulka 2.

Na následujícím obrázku 9 je uveden formát zprávy v binární podobě.

Tabulka 2 – Význam polí ve zprávách mobilní zařízení — server.

atribut	hodnota	popis
length	int	Délka zprávy.
messageType	int DEVICE_MESSAGE(0) ID(1) NOTHING_TO_SEND(126) UNKNOWN(127)	Typ zprávy. Zpráva pocházející z komunikačního zařízení. Jedná se o zprávu z komunikace mezi mobilním zařízením a komunikačním zařízením, která je pouze přeposlána na server. Zpráva s identifikátorem mobilního zařízení. Typ zprávy pro označení, že už není nic k odeslání a komunikace proběhla v pořádku. Neznámý typ zprávy, pravděpodobně neimplementovaný.
date	long	Datum zprávy.
data	byte[]	Tělo zprávy — binární data.



Obrázek 9 – Binární formát zpráv mobilní zařízení — komunikační zařízení.

Komunikace mobilního zařízení se serverem probíhá následujícím způsobem:

1. Mobilní aplikace naváže spojení se serverem.
2. Jako první posílá mobilní aplikace zprávu se svým identifikátorem. Pokud žádný nemá přiřazen od serveru, pošle zprávu prázdnou a server jí identifikátor přiřadí a pošle v odpovědi.
3. Mobilní aplikace posílá zprávy na server a čeká na potvrzení o zpracování.
4. Pokud operace neproběhne v pořádku, zprávy se budou odesílat znovu při příštím odesílání, jinak jsou odstraněny z fronty odesílání.

3.3 Databáze mobilního zařízení

Tato sekce se věnuje popisu databáze na mobilním zařízení. Databáze je využita hned pro několik činností. Zejména pro ukládání naměřených hodnot a také aktuální konfigurace — například počtu a typu senzorů, jejich náležitosti k měřicím zařízením a podobně. Také samozřejmě slouží pro ukládání zpráv, které se odesílají na server. V neposlední řadě je také využita pro servisní účely — uchovává poslední zprávy, které si vyměnila s komunikačním zařízením. Databáze v zařízení Android je specifická tím, že není přístupná ke správě z vnějšku aplikace, ale kód pro vytváření a manipulaci s databázovými objekty musí být přítomen přímo v aplikaci. To vede k tomu, že je většina příkazů SQL „natvrdo“ zapsána ve zdrojových kódech Javy. Z toho důvodu také každou tabulku reprezentuje jedna Java třída.

Pro splnění výše uvedených úkolů databáze obsahuje čtyři tabulky a jeden pohled (zapsány ve tvaru „*název_tabulky (název Java třídy)*“):

- tabulku *device_messages* (*DeviceMessages*),
- tabulku *measurements* (*Measurements*),
- tabulku *measured_values* (*Values*),
- pohled *measurement_values* (*ViewMeasurementValues*),
- tabulku *server_messages* (*ServerMessages*).

Tyto databázové objekty budou samostatně popsány v následujících podkapitolách. V příloze E je zobrazen model databáze mobilního zařízení.

3.3.1 Tabulka device_messages

Tato tabulka slouží pro uchování zpráv, které si mobilní aplikace vyměňuje s komunikačním zařízením. Toho se využívá zejména v servisní části, kde lze sledovat odeslané a přijaté zprávy. SQL kód pro popis tabulky je uveden v následující ukázce.

```
CREATE TABLE device_messages
(
    -- Umělý primární klíč pro každou zprávu
    _id INTEGER PRIMARY KEY,
    -- Datum vzniku zprávy
    date INTEGER,
    -- Typ zprávy
    message_type INTEGER,
    -- Objekt zprávy
    message_object INTEGER,
    -- Binární data zprávy
    message BLOB
)
```

3.3.2 Tabulka measurements

V této tabulce jsou uchovávány informace o jednotlivých měřeních. Zde jsou uchovány pouze informace o čase měření a jednotlivé hodnoty jsou v samostatné tabulce. To poskytuje flexibilitu databáze pro různé změny konfigurace bez potřeby měnit strukturu databáze. SQL kód je uveden v následující ukázce.

```
CREATE TABLE measurements
(
    -- Umělý primární klíč pro každé měření
    _id INTEGER PRIMARY KEY,
    -- Identifikátor měřicího zařízení
    device_id INTEGER,
    -- Datum měření
    date INTEGER
)
```

3.3.3 Tabulka *measured_values*

Tabulka *measured_values* obsahuje hodnoty, které náleží k jednotlivým měřením uvedeným v tabulce *measurements*. Jelikož tabulka *measurements* již specifikuje, ze kterého měřicího zařízení měření pochází, je u každého záznamu v tabulce *measured_values* uchováván pouze identifikátor senzoru a identifikátor měření z tabulky *measurements*.

```
CREATE TABLE measured_values
(
    -- ID záznamu z tabulky measurements
    id_measurements INTEGER,
    -- Identifikátor senzoru
    sensor_id INTEGER,
    -- Naměřená hodnota
    value REAL,
    FOREIGN KEY(id_measurements) REFERENCES
        measurements(_id) ON DELETE CASCADE ON UPDATE
        CASCADE,
    PRIMARY KEY(id_measurements, sensor_id)
)
```

3.3.4 Pohled *measurement_values*

Tento pohled je jediným pohledem v databázi. Nabízí komplexní náhled na naměřené hodnoty. Obsahuje data z tabulek *measurements* a *measured_values*, která jsou spojena. Výsledkem je relace obsahující n-tice, ve kterých je údaj o ID měřicího zařízení, ID senzoru, naměřená hodnota a datum.

```
CREATE VIEW measurement_values AS
SELECT
    id_measurements, device_id, sensor_id, value, date
FROM measurements
JOIN measured_values ON _id = id_measurements
```

3.3.5 Tabulka `server_messages`

V tabulce `server_messages` jsou uchovávány zprávy určené k odeslání na server. Obsahuje data o času zprávy, typu zprávy a datech zprávy. Po úspěšném odeslání jsou příslušné zprávy z této tabulky smazány.

```
CREATE TABLE server_messages
(
    -- Umělý primární klíč pro každou zprávu k odeslání
    _id INTEGER PRIMARY KEY,
    -- Datum vzniku zprávy
    date INTEGER,
    -- Typ zprávy
    message_type INTEGER,
    -- Binární data zprávy
    message BLOB
)
```

3.4 Databáze serveru

Tato kapitola popisuje databázi nasazenou na serveru, která se skládá ze dvou schémat:

- measurement,
 - Schéma pro databázové objekty týkající se naměřených hodnot.
- device_configurations.
 - Schéma pro databázové objekty týkající se mobilních zařízení a jejich konfigurací. Tyto databázové objekty využívá technologie JPA.

V následujících podkapitolách budou popsány nejdůležitější databázové objekty. Jména jednotlivých databázových objektů v názvech podkapitol odkazují přímo na schémata, ze kterých pocházejí. Jsou totiž ve formátu *schéma.objekt*. V příloze F je k vidění databázový model reprezentující serverovou databázi.

3.4.1 Tabulka measurement.measurements

V této tabulce jsou uchovány informace o jednotlivých proběhlých měřeních. Tabulka je z hlediska účelu obdobná jako tabulka *measurements* na mobilním zařízení. Konkrétní naměřené hodnoty jsou opět v samostatné tabulce.

```
CREATE TABLE measurement.measurements(  
  -- Umělý primární klíč pro každé měření  
  id bigserial NOT NULL,  
  -- Identifikátor mobilního zařízení na kterém bylo  
  měřeno  
  android_id integer NOT NULL,  
  -- Identifikátor příslušného měřicího zařízení  
  device_id smallint NOT NULL,  
  -- Datum měření  
  measurement_date timestamp NOT NULL,  
  CONSTRAINT pk_measurements PRIMARY KEY (id)  
);
```

3.4.2 Tabulka measurement.measured_values

Tabulka s jednotlivými hodnotami měření. Jedná se o obdobu tabulky *measured_values* na mobilním zařízení.

```
CREATE TABLE measurement.measured_values(  
  -- Identifikátor záznamu v tabulce  
  measurement.measurements  
  id_measurement bigint,  
  -- Identifikátor senzoru  
  sensor_id smallint NOT NULL,  
  -- Naměřená hodnota  
  measured_value numeric(5,2)  
);
```

3.4.3 Pohled measurement.measurement_values

Pohled na spojené tabulky *measurement.measurements* a *measurement.measurement_values*. Je obdobou pohledu *measurement_values* z mobilní aplikace.

```
CREATE VIEW measurement.measurement_values
AS WITH myView AS
(
SELECT
    id, android_id, device_id, sensor_id, measured_value,
    measurement_date
FROM
    measurement.measurements
JOIN
    measurement.measured_values ON measurements.id =
    measured_values.id_measurement
)
SELECT id, android_id, device_id, sensor_id,
    measured_value, measurement_date
FROM myView;
```

3.4.4 Tabulka device_configurations.androids

První tabulkou týkající se konfigurace zařízení je tabulka *device_configurations.androids*. Ta obsahuje identifikátory jednotlivých mobilních zařízení, které komunikují nebo v minulosti komunikovaly se serverem. Každý záznam jednoznačně určuje mobilní zařízení i jeho konfiguraci. Je možné a dokonce i pravděpodobné, že jedno fyzické mobilní zařízení bude mít v této tabulce více záznamů z důvodu různých konfigurací. Pokud totiž mobilní zařízení změní svoji konfiguraci, změní se i jeho identifikátor na serveru. Takto lze dohledat i historické údaje o konfiguracích.

```
CREATE TABLE device_configurations.androids(
    -- Identifikátor mobilního zařízení a jeho konfigurace
    id integer NOT NULL,
    CONSTRAINT pk_androids PRIMARY KEY (id)
);
```

3.4.5 Tabulka `device_configurations.devices`

Tato tabulka obsahuje záznamy o jednotlivých měřicích zařízeních příslušných k mobilním zařízením z tabulky `device_configurations.androids`.

```
CREATE TABLE device_configurations.devices(  
    -- Identifikátor měřicího zařízení v rámci mobilního  
    -- zařízení  
    id smallint NOT NULL,  
    -- Identifikátor konfigurace mobilního zařízení  
    android_id integer NOT NULL,  
    CONSTRAINT pk_devices PRIMARY KEY (android_id,id)  
);
```

3.4.6 Tabulka `device_configurations.sensors`

V této tabulce jsou uchovány údaje o jednotlivých senzorech, které přísluší k měřicím zařízením z tabulky `device_configurations.devices`.

```
CREATE TABLE device_configurations.sensors(  
    -- Identifikátor senzoru v měřicím zařízení  
    id smallint NOT NULL,  
    -- Typ senzoru  
    value_type varchar(20),  
    -- Identifikátor příslušného měřicího zařízení  
    parent_device smallint NOT NULL,  
    -- Identifikátor příslušného mobilního zařízení  
    parent_android integer NOT NULL,  
    CONSTRAINT pk_sensors PRIMARY KEY  
        (id,parent_device,parent_android)  
);
```

3.4.7 Tabulka `device_configurations.limits`

Tabulka `device_configurations.limits` obsahuje údaje o limitech pro jednotlivé senzory. Ve webové aplikaci lze pro senzory nastavit limity, které nesmí jejich naměřené hodnoty překročit. Každý limit má samostatný identifikátor a datum od kdy platí, jelikož jeden senzor může mít v různých časech jinak nastavené limity.

```
CREATE TABLE device_configurations.limits(  
    -- Umělý primární klíč pro každý limit  
    id_limit bigserial NOT NULL,  
    -- Identifikátor mobilního zařízení  
    id_android integer,  
    -- Identifikátor měřicího zařízení  
    id_device smallint,  
    -- Identifikátor senzoru  
    id_sensor smallint,  
    -- Horní hodnota limitu  
    upper_value numeric(5,2),  
    -- Spodní hodnota limitu  
    lower_value numeric(5,2),  
    -- Datum od kdy platí limit  
    date_set timestamp,  
    CONSTRAINT pk_limits PRIMARY KEY (id_limit)  
);
```

4 Mobilní aplikace

Tato kapitola se věnuje představení aplikace pro mobilní zařízení. V následujících podkapitolách bude aplikace popsána z hlediska její implementace a použití. Nejprve však následuje malá rekapitulace, co všechno má ve své kompetenci.

Mobilní aplikace má za úkol komunikovat se zařízeními, která slouží k měření různých hodnot. Takto získané informace uchovává ve své databázi a zobrazuje uživateli ve formě obrazovky s připojenými měřicími zařízeními a grafů s naměřenými hodnotami. Také komunikuje se serverem, kam odesílá relevantní informace pro další vzdálené zpracování.

Jedná se o aplikaci napsanou v programovacím jazyce Java verze 6. Tato verze je použita z důvodu zpětné kompatibility až do verze OS Android Gingerbread (2.3). V aplikaci je také využito mnoho dalších technologií, které byly uvedeny v předchozích kapitolách. Jedná se zejména o kapitolu o použitých technologiích a podkapitolu Mobilní aplikace.

4.1 Návrh a implementace mobilní aplikace

Platforma Android je velice restriktivní v ohledu návrhu aplikace. Obsahuje mnoho velmi specifických omezení a občas i úskalí, která nejsou na platformě Java běžně přítomna (alespoň v desktopové verzi). To je samozřejmě způsobeno specifickými nároky na provoz softwaru na mobilních zařízeních obecně. V následujícím textu bude uvedeno několik příkladů, kdy se vývoj pro platformu Android velmi liší od vývoje pro desktopovou platformu a co tyto odlišnosti znamenají pro mobilní aplikaci.

Životní cyklus aplikací není plně v rukou programátora, ale je silně řízen systémem Android. Operační systém se může kdykoliv rozhodnout ukončit běžící aplikaci například z důvodu nedostatku operační paměti v zařízení. Proto se musí vždy počítat s tím, že aplikace může být kdykoliv během svého běhu přerušena. Nezanedbatelný je i vliv takzvaných konfiguračních změn, kdy může být grafická část aktuální aplikace přerušena z důvodu otočení zařízení a tím pádem i otočení obrazovky kupříkladu do režimu landscape. Takováto změna konfigurace, jak již bylo zmíněno, vede k tomu, že je aktuální část programu s grafickým rozhraním vypnuta a znovu zapnuta. Programátor tak musí v jistých případech ručně uchovat stav obrazovky před otočením a po restartu tento stav znovu obnovit.

Další velmi specifickou záležitostí platformy Android je, že aplikace jsou velmi orientované na grafické uživatelské rozhraní. Logika většiny mobilních aplikací je obsažena právě v práci s grafickým rozhraním. Pokud to není viditelné, aplikace většinou neběží, což převážně většině aplikací vůbec nevádí. Pokud uživatel s aplikací právě nepracuje, není důvod aby běžela a čerpala poměrně omezené prostředky mobilního zařízení. Avšak mobilní aplikace v této práci daleko více pracuje na pozadí než interaguje s uživatelem. Naštěstí i pro tento případ je platforma Android připravena a nabízí možnost provádění logiky aplikací v takzvaných službách. To jsou procesy, které běží na pozadí a například jen zřídka reagují s uživatelem. Už z podstaty mobilní aplikace v této práci je jasné, že právě tyto služby budou často pro její běh používány.

Hlavními komponentami aplikace jsou takzvané aktivity (jednotlivé obrazovky), které představují grafické uživatelské rozhraní, a dvě hlavní služby:

- Systém aktivit,
 - GUI aplikace skládající se z jednotlivých obrazovek, mezi kterými uživatel naviguje dle libosti.
- *DeviceCommunicationService*,
 - Služba pro komunikaci s komunikačními a potažmo s měřicími zařízeními.
- *ServerCommunicationService*.
 - Služba pro komunikaci se serverem.

V dalších podkapitolách jsou podrobněji představeny jednotlivé služby a uvedeny problémy, se kterými bylo potřeba se při tvorbě aplikace vypořádat. Grafické rozhraní bude popsáno v následující kapitole, věnující se představení vytvořené aplikace.

4.1.1 Služba DeviceCommunicationService

Služba pro komunikaci s komunikačním zařízením se rozeběhne ve chvíli, kdy uživatel zvolí možnost připojení ke komunikačnímu zařízení. V aplikaci je sice pouze možnost spojení s fiktivním (dummy) zařízením na popud uživatele, avšak není problém přidat například zařízení, se kterým se začne komunikovat, když se detekuje jeho připojení přes USB. Pro přidání dalších implementací komunikačních zařízení stačí programovat proti připraveným rozhraním, která obsahují metody pro řízení životního cyklu spojení s komunikačním zařízením. Za nejdůležitější rozhraní lze považovat *MeasurementCommunicator*, který slouží ke správě komunikace s komunikačním zařízením a jehož kód vypadá následovně.

```
public interface MeasurementCommunicator extends
    OutputToDevice, MessageHandler {
    void connect() throws IOException;
    void disconnect() throws IOException;
    MessageDispatcher getMessageDispatcher();
}
```

Pro úplnost jsou ještě uvedena rozhraní, která *MeasurementCommunicator* implementuje. *OutputToDevice* specifikuje metodu pro odchozí komunikaci s komunikačním zařízením. Jak je patrné z ukázky, rozhraní je velmi jednoduché. Obsahuje jedinou metodu, která slouží k odeslání zprávy.

```
public interface OutputToDevice {
    void sendMessage(MessageMeric message) throws
        IOException;
}
```

Do jisté míry opačným rozhraním k předchozímu je *MessageHandler*, který naopak slouží ke zpracování příchozích zpráv. Opět má pouze jednu jedinou metodu, která slouží ke zpracování zprávy.

```
public interface MessageHandler {  
    void handle(MessageMeric message) throws Exception;  
}
```

Nyní zpět k popisu metod rozhraní *MeasurementCommunicator*. Názvy metod *connect()* a *disconnect()* mluví samy za sebe. Slouží k připojení a odpojení měřicího zařízení. Metodou, která potřebuje více rozepsat, je *getMessageDispatcher*. Ta má za úkol navracení objektu, který slouží ke zpracování příchozích zpráv. S přibývajícím počtem typů zpráv a kódu pro jejich zpracování by bylo velmi nepraktické řešit jejich zpracování na jednom místě. Proto byla vytvořena třída *MessageDispatcher*, která tuto práci obstarává. Skrze metodu *registerHandler()* se dvěma parametry — *handler* typu *MessageHandler* a *predicate* typu *Predicate<MessageMeric>* — lze registrovat handler pro zpracování zpráv. Ten bude použit, pokud příchozí zpráva vyhovuje podmínce reprezentované parametrem *predicate*. Při příjmu nové zprávy se tedy na zprávu aplikují filtry typu *Predicate<MessageMeric>* a volají se metody příslušných handlerů, u kterých filtr navrátí hodnotu *true*. Navíc je metoda zmíněných handlerů spouštěna na UI vlákne, takže lze například zobrazovat různé notifikace a klasicky pracovat s GUI. Zpracování však nesmí trvat příliš dlouho, aby nedošlo k „zamrznutí“ grafického rozhraní. Pro dlouho trvající činnosti by musel každý handler použít samostatné vlákno pro zpracování zprávy na pozadí.

Implementace třídy pro reprezentaci komunikačního zařízení již není ničím omezena. Z důvodu co největší svobody pro implementaci bylo použito rozhraní pouze pro komunikátor — *MeasurementCommunicator*. Díky tomu je možné například v implementaci samotného zařízení vytvořit smyčku pro komunikaci s USB zařízením, která serializuje zprávy do binární podoby pro odeslání do výstupního proudu a naopak na vstupním proudu naslouchá a deserializuje binární data na zprávy, které poté předává ke zpracování právě komunikátoru — instanci *MeasurementCommunicator*. Jak probíhá komunikace a interakce mezi třídami je možné vidět na obrázku 5.

4.1.2 Služba *ServerCommunicationService*

Služba pro odesílání dat na server pracuje poněkud jiným způsobem než služba pro komunikaci s komunikačním zařízením. Systém Android nabízí programátorům k využití systémovou službu *AlarmManager*. U ní si lze zaregistrovat opakované zasílání zpráv v libovolném intervalu nebo i jednorázové zaslání. Výhodou využití této systémové služby je to, že optimalizuje energetickou spotřebu mobilního zařízení. Dokáže se například aktivovat i když je CPU v režimu spánku a během vykonávání metody pro příjem zprávy (v naslouchající instanci příslušné třídy) drží zámek pro uspání CPU (*wake lock*), takže je možné po tuto dobu vykonávat instrukce aplikace. Mimo jiné umí naplánovat odeslání více zaregistrovaných zpráv v dávce najednou, čímž minimalizuje počet probuzení procesoru (GOOGLE, 2015a). Zasílanými zprávami jsou instance třídy *PendingIntent*, které nesou údaje o akci, kterou představují (tato akce je nastavena při

registraci v *AlarmManager*). Obsahují také zpravidla identifikaci třídy, která slouží pro jejich příjem. Tato třída musí být uvedena v manifestu *AndroidManifest.xml*, takže se jedná o statickou konfiguraci příjmu zpráv. Kvůli tomu nelze tyto naslouchací třídy volit během provádění programu. Třída *PendingIntent* je obdoba třídy *Intent*, která se v systému Android používá především pro komunikaci mezi samostatnými částmi v rámci jedné aplikace nebo i pro komunikaci mezi více aplikacemi. Dalo by se říci, že se jedná v podstatě o prostředek IPC. Při navázání spojení s komunikačním zařízením se tedy zaregistruje zpráva, která se v požadovaném intervalu posílá naslouchající třídě *ServerSenderBCReceiver*, která při přijetí zprávy spustí jedno kolo komunikace se serverem. Při něm se odešlou všechny zprávy ve frontě k odeslání.

Samotné odesílání na server za použití služby *AlarmManager* však obsahuje některá úskalí. Jak již bylo zmíněno, CPU zařízení může být v režimu spánku. Služba *AlarmManager* proto získá zámek CPU během zpracování zprávy v naslouchající třídě. Problémem je však, že v naslouchající třídě se pouze asynchronně spustí služba pro odesílání na server. Může tedy dojít k tomu, že se CPU uspí dříve než se vůbec služba spustí. Navíc může být stejným způsobem uspána Wi-Fi, pokud uživatel nějakou chvíli zařízení aktivně nepoužívá (GOOGLE, 2015b). Proto je potřeba před spuštěním služby získat zámky pro CPU i Wi-Fi a po ukončení služby je opět uvolnit. Hlavní kód služby pro komunikaci se serverem je v následující ukázce.

Metoda *doWakefulWork()* vykonává kód na samostatném vlákně díky využití třídy *AsyncTask* z Androidu. Při jejím běhu je držen zámek CPU i Wi-Fi. V této metodě je vytvořena instance třídy *ServerSender* a zavolána její metoda *send()*, která provede odeslání zpráv ve frontě na server.

```

@Override
public int onStartCommand(Intent intent, int flags, int
    startId) {
    if (!running) {
        running = true;
        final PowerManager.WakeLock wakeLock =
            getWakeLock(this
                .getApplicationContext());
        final WifiManager.WifiLock wifiLock =
            getWifiLock(this
                .getApplicationContext());
        if (!wakeLock.isHeld()) {
            wakeLock.acquire();
        }
        if (!wifiLock.isHeld()) {
            wifiLock.acquire();
        }
        (new AsyncTask<Intent, Void, Void>() {
            @Override
            protected Void doInBackground(Intent...
                params) {
                doWakefulWork(params[0]);
                return null;
            }
            @Override
            protected void onPostExecute(Void result) {
                running = false;
                if (wakeLock.isHeld()) {
                    wakeLock.release();
                }
                if (wifiLock.isHeld()) {
                    wifiLock.release();
                }
            }
        }).execute(intent);
    }
    return START_NOT_STICKY;
}

```

Metoda provádějící logiku služby *ServerCommunicationService*.

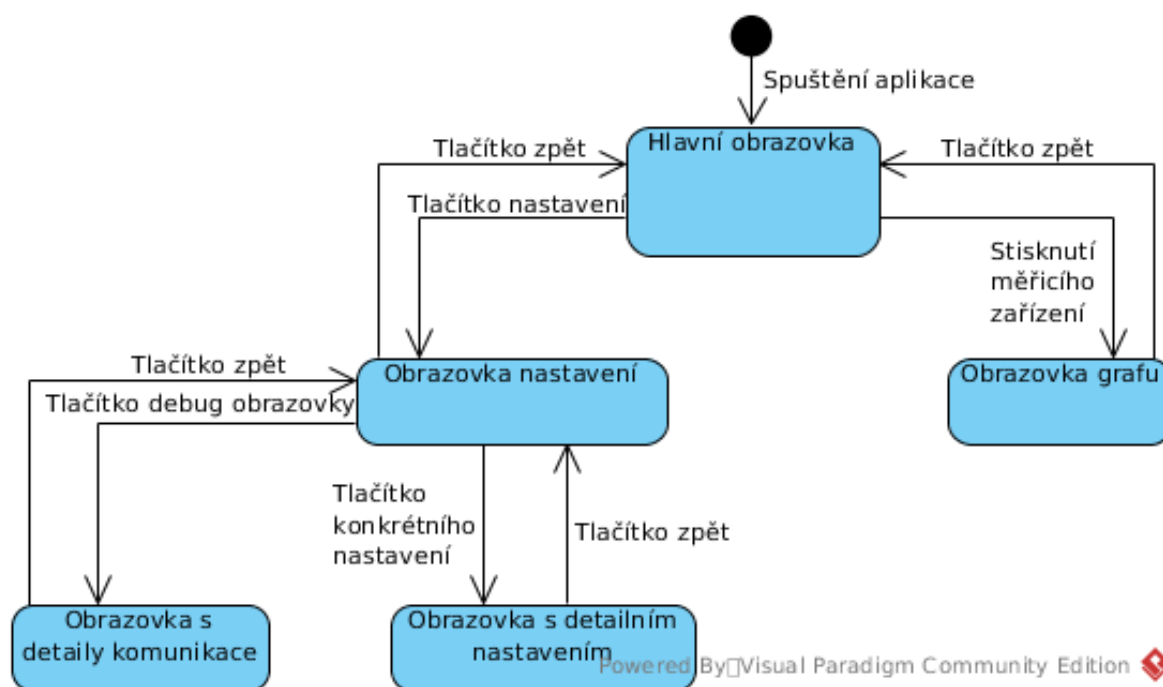
4.2 Předvedení mobilní aplikace

Tato kapitola se zabývá představením mobilní aplikace z hlediska uživatele. Slouží především k rychlé kontrole posledních naměřených hodnot a k jejich automatickému odesílání na server.

Použití aplikace je tedy následující:

1. Uživatel spustí aplikaci, čímž se dostane na hlavní obrazovku.
2. Uživatel stiskne tlačítko pro spojení s komunikačním zařízením.
3. Je spuštěna služba pro komunikaci s komunikačním zařízením.
4. Služba úspěšně naváže spojení s komunikačním zařízením a aktivuje se služba pro odesílání dat na server.

Na následujícím obrázku je ukázán stavový diagram, který reprezentuje navigaci v grafickém rozhraní mobilní aplikace. Zobrazuje hlavní obrazovky aplikace a přechody mezi nimi.



Obrázek 10 – Navigace v GUI mobilní aplikace.

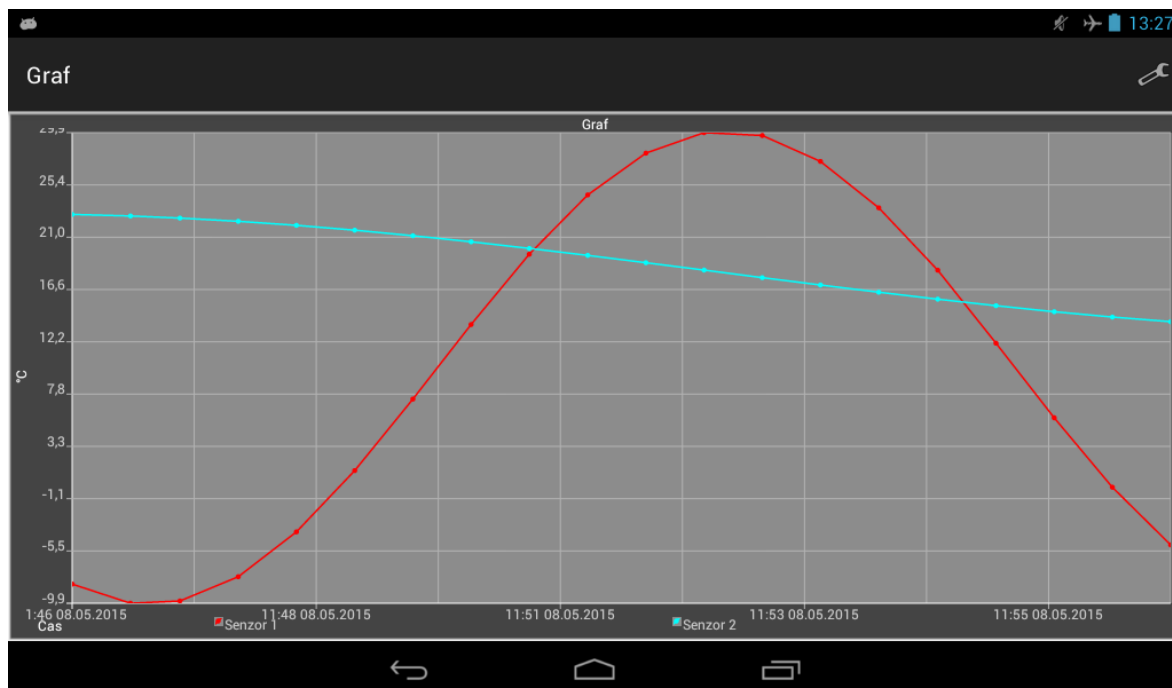
Hlavní obrazovka aplikace je vidět na následujícím obrázku 11. Slouží k přehlednému zobrazení připojených měřicích zařízení a jejich aktuálních hodnot. Ve skutečnosti je na této obrazovce pouze komponenta webového prohlížeče, která zobrazuje HTML stránku. Na té jsou jednotlivá měřicí zařízení reprezentována komponentami napsanými v JavaScriptu. To umožňuje jednotný vzhled jak v mobilní aplikaci, tak ve webové aplikaci, která bude představena v dalších kapitolách. Vzhled celé stránky i komponent lze měnit pomocí CSS souborů nebo úprav v souborech s JavaScriptem. Tímto způsobem

bylo možné se vyhnout problémům, kdy by pro Android nativně napsané komponenty mohly vypadat různě na různých mobilních zařízeních.



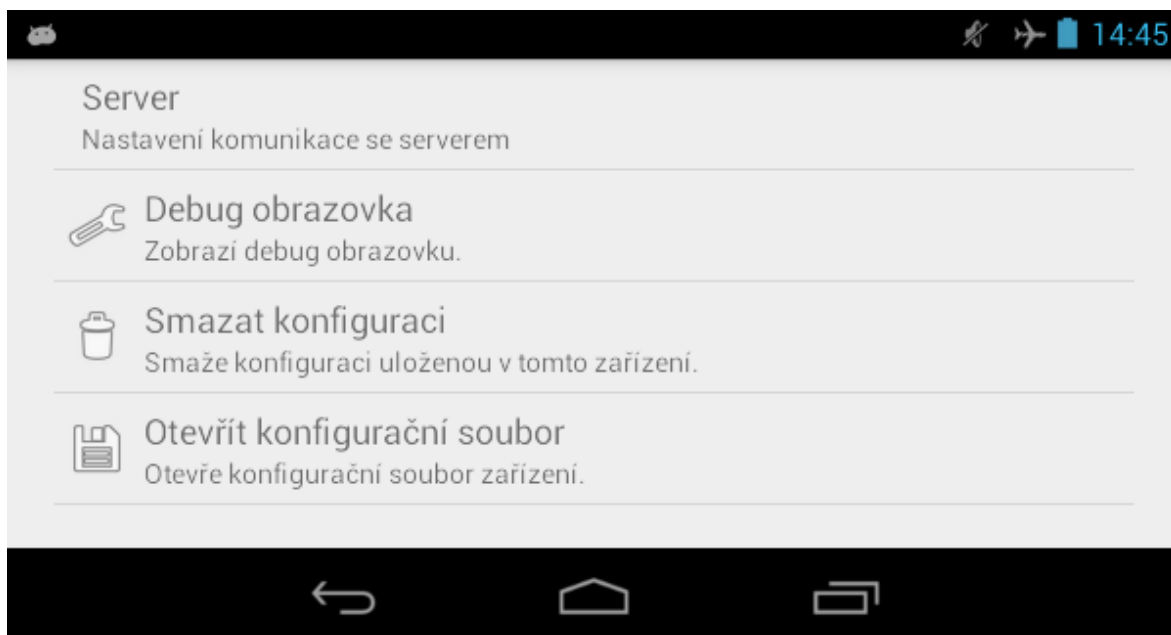
Obrázek 11 – Hlavní obrazovka mobilní aplikace.

Při „kliknutí“ na zvolené měřicí zařízení se otevře obrazovka s příslušným grafem. Ta je k vidění na obrázku 12.

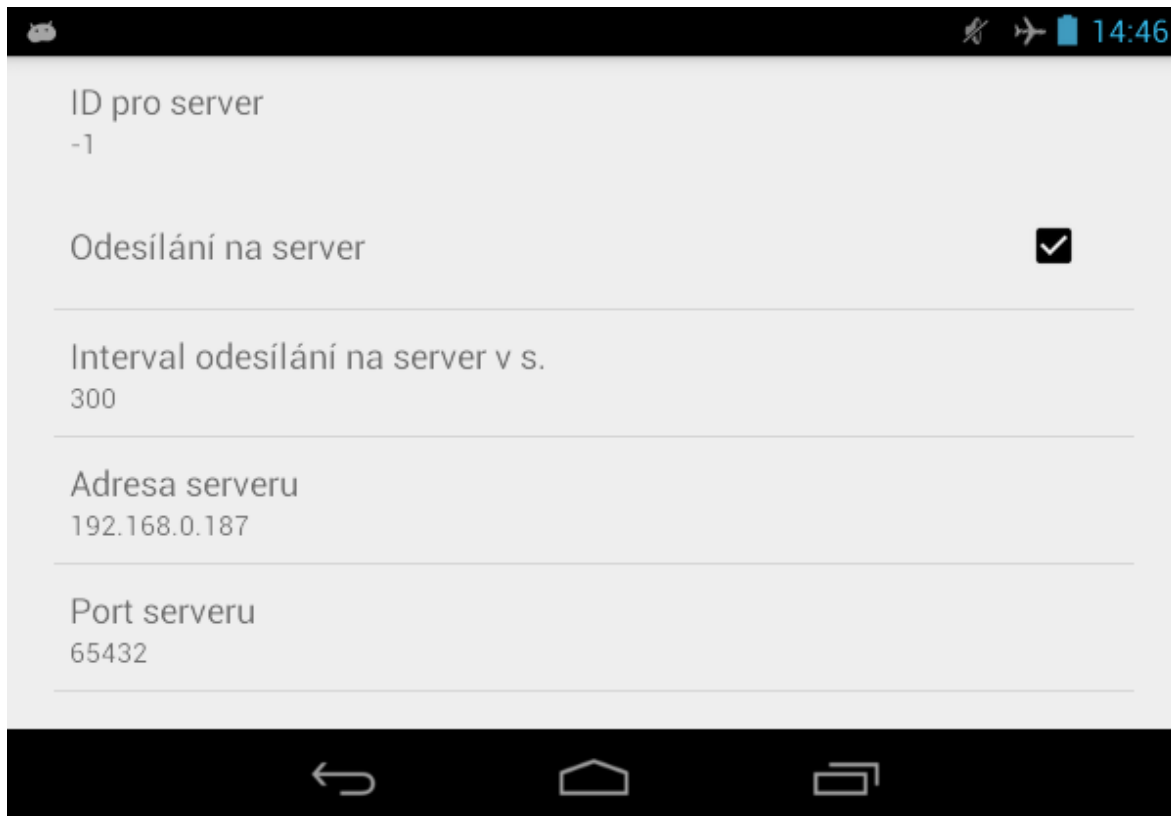


Obrázek 12 – Zobrazení grafu v mobilní aplikaci.

Důležitou obrazovkou aplikace je i obrazovka nastavení, kterou lze vidět na obrázku 13. Ta obsahuje možnost zobrazení nastavení komunikace se serverem, jehož příslušná obrazovka je na obrázku 14.

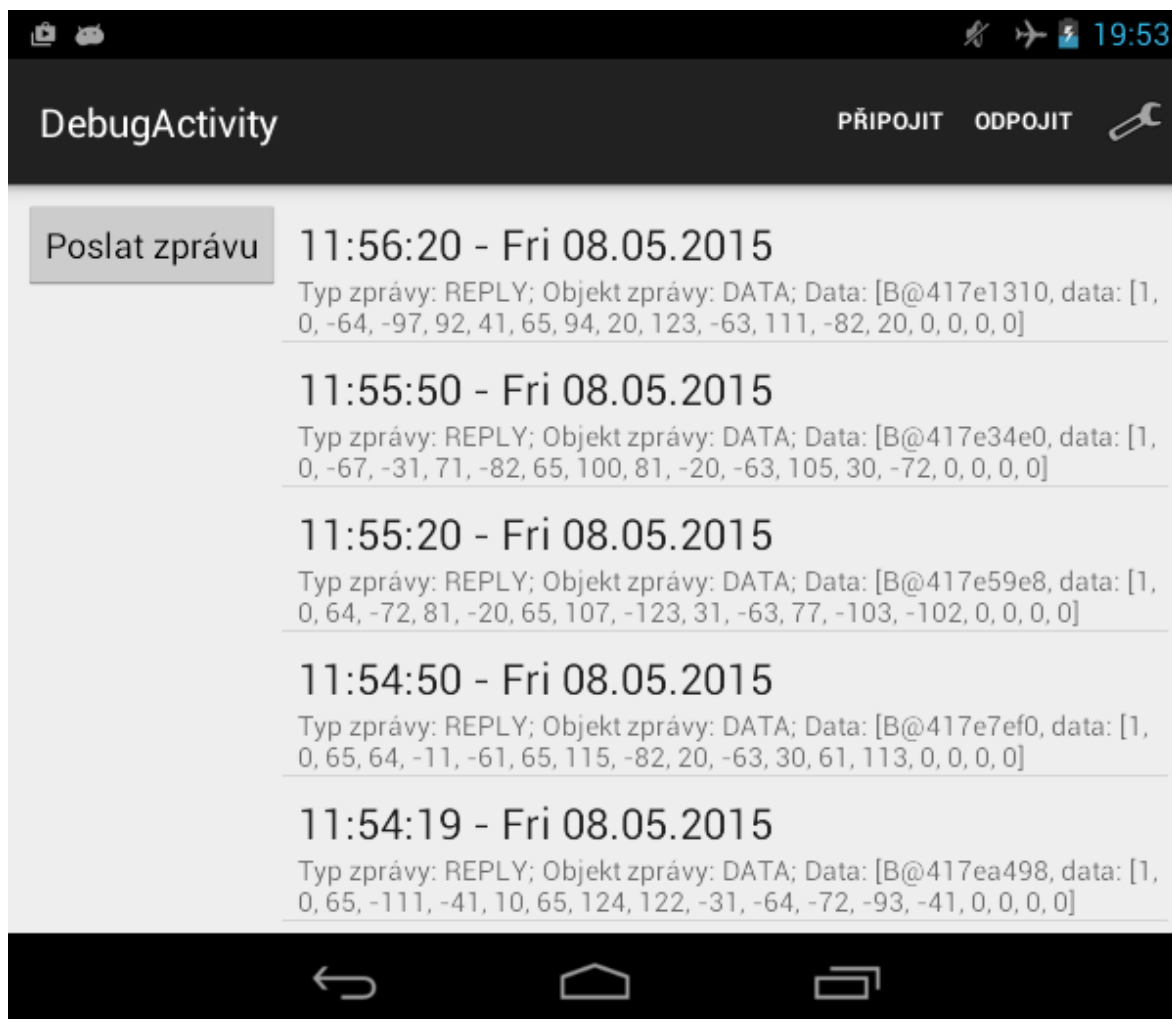


Obrázek 13 – Obrazovka nastavení mobilní aplikace.



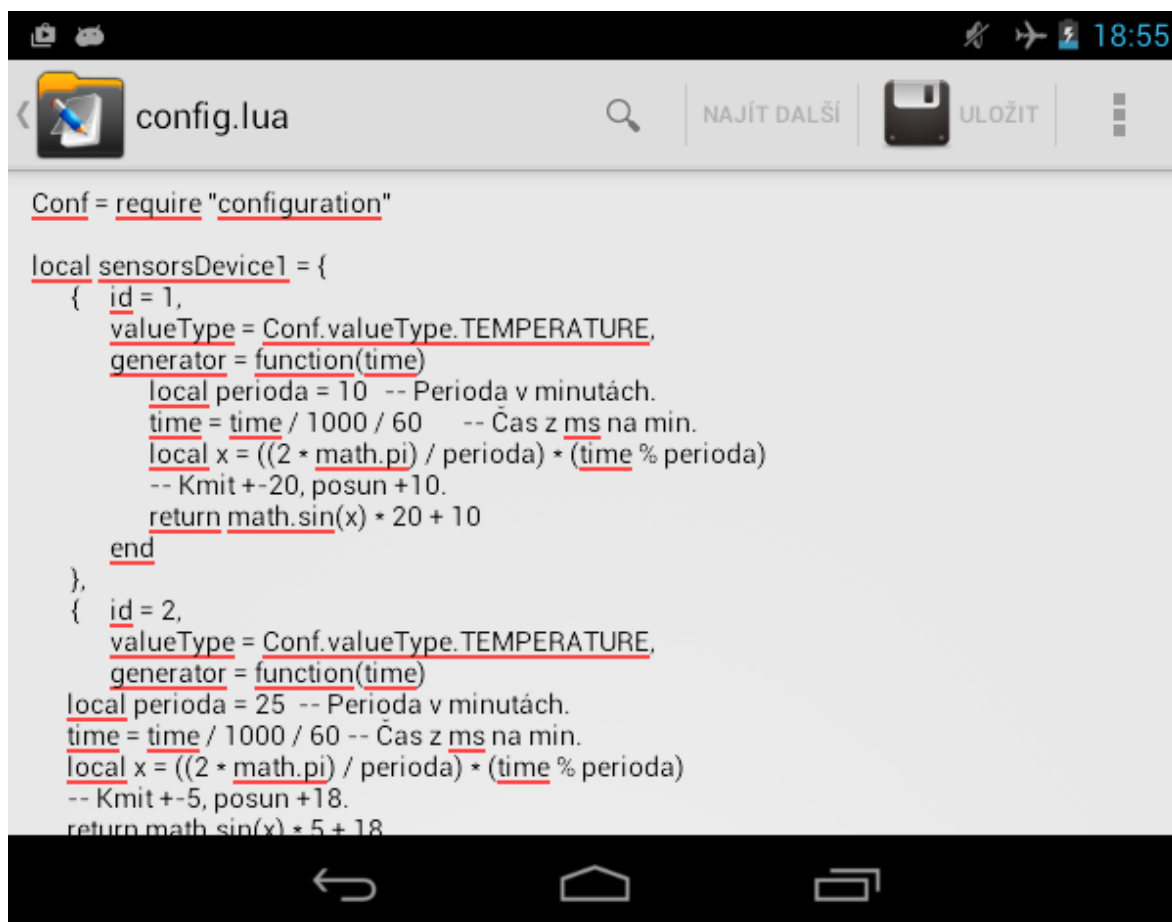
Obrázek 14 – Obrazovka nastavení komunikace se serverem v mobilní aplikaci.

Pro účely údržby a ladění aplikace je k dispozici obrazovka, která uživateli poskytuje možnost sledování komunikace mezi mobilním a komunikačním zařízením. Do této obrazovky se uživatel dostane stisknutím tlačítka „Debug obrazovka“, které se nachází v nastavení. Zde je zobrazen seznam příchozích a odchozích zpráv seřazený dle času. K dispozici je i tlačítko, kterým je možné ručně odeslat libovolnou zprávu komunikačnímu zařízení. Ukázka celé obrazovky komunikace je na obrázku 15.



Obrázek 15 – Obrazovka s komunikací mobilního a komunikačního zařízení.

Také lze smazat všechna uložená data v databázi a identifikační číslo mobilní aplikace pomocí tlačítka „Smazat konfiguraci“. Tlačítko „Otevřít konfigurační soubor“ slouží k otevření konfiguračního souboru fiktivního komunikačního zařízení v jazyce Lua v externím textovém editoru. Ten musí být na zařízení dodatečně nainstalován, pokud je potřeba tento soubor zobrazit nebo editovat. Jak taková editace následně vypadá je znázorněno na obrázku 16, kde je k editaci použit program X-plore.



Obrázek 16 – Zobrazení konfiguračního souboru externím textovým editorem.

5 Serverová aplikace

V této kapitole je popsána aplikace pro server. Bude připomenut účel aplikace a v následujících kapitolách postupně uveden její návrh a implementace.

Úkolem serverové aplikace je přijímat zprávy z mobilní aplikace a tyto zprávy zpracovávat a uchovávat. Jelikož je potřeba zpracovávat zprávy z mnoha zařízení najednou, je důležitá otázka výkonu vstupní části programu. Z tohoto důvodu byla zvolena technologie Java NIO pro neblokující práci se vstupem/výstupem, která by měla zajistit dostatečný výkon při minimálním zatížení serveru. Použitím této technologie se lze vyhnout vytváření velkého množství vláken, avšak za cenu zvýšené náročnosti návrhu a implementace vstupně/výstupní části aplikace. Pro neblokující komunikaci s mobilními zařízeními je potřeba vytvořit stavový automat. Ten poté přepíná stavy pro jednotlivá spojení podle toho, jestli je potřeba data číst nebo zapisovat.

V programu je také pro ladění použito Java Logging API. Díky tomu lze centrálně řídit výpis ladicích informací — například od jaké úrovně upozornění se budou informace logovat (například úroveň varování a vyšší) nebo jakým způsobem. Není problém jednoduše nastavit výpis do konzole nebo do souboru. Dokonce lze konfigurovat formát výpisu například do HTML nebo XML. Další použité technologie jsou uvedeny v předchozích kapitolách, které se tomuto tématu věnují, zejména v podkapitole Serverová komunikační aplikace kapitoly pojednávající o použitých technologiích.

5.1 Návrh a implementace serverové aplikace

Při popisu serverové aplikace bude soustředěna pozornost na dvě hlavní funkčnosti aplikace. V první řadě půjde o komunikaci s mobilním zařízením, zejména o vstupně/výstupní část aplikace. Poté bude probráno zpracování zpráv příchozích z mobilních zařízení.

5.1.1 Vstupně/výstupní část

Za srdce celé serverové aplikace lze považovat tu část programu, která slouží k obsluze spojení s mobilními zařízeními. Jelikož se jedná o nejvíce zatíženou část aplikace, která by měla zvládnout obsloužit mnoho připojených mobilních zařízení, byla zvolena technologie Java NIO. Díky tomu, že byl použit neblokující vstup/výstup, má hlavní smyčku pro obsluhu všech otevřených spojení na starost jedno vlákno. Proto bylo potřeba vytvořit stavový automat, který podle aktuálního stavu provádí příjem nových spojení nebo čtení/zápis z/do spojení navázaných. Tento automat je řízený frontou příkazů, které ovládají stavy jednotlivých kanálů = navázaných spojení. Jelikož server nezačíná komunikovat jako první, výchozím stavem spojení je režim naslouchání. Celý postup komunikace je následující:

1. Server přijme příchozí spojení z mobilního zařízení a nastaví otevřený kanál pro režim naslouchání.

2. Server postupně přijímá a zpracovává zprávy ze zařízení.
3. Server chce odpovědět na zprávu. Proto přidá do fronty stavového automatu požadavek na změnu stavu kanálu do režimu pro zápis a data, která se mají zapsat.
4. Při vykonávání hlavní smyčky komunikace se zjistí, že je kanál připravený pro zápis. Vyzvednou se data k odeslání a odešlou se. Poté se nastaví kanál zpět do režimu naslouchání.
5. Po úspěšném zpracování všech zpráv, pokud již server nepotřebuje komunikovat s mobilním zařízením, odešle zprávu `NOTHING_TO_SEND(126)`. Tím pádem ukončí mobilní zařízení komunikaci.

Při čtení se přijatá data uloží do bufferu NIO. Z něj se data přkopírují do samostatného pole bytů, které se předá ke zpracování pomocnému vláknu — aby nebylo zbytečně brzděno vlákno komunikační. Příslušný kód je možné vidět na následující ukázce.

```
private void read(SelectionKey key) throws IOException {
    SocketChannel socketChannel = (SocketChannel)
        key.channel();

    while (true) {
        readBuffer.clear();
        int numRead = socketChannel.read(readBuffer);

        if (numRead == -1) {
            // Druhá strana uzavřela spojení.
            LOGGER.log(Level.FINE, "Connection closed by
                {0}",
                    socketChannel.getRemoteAddress());
            removeKey(key);
            return;
        } else if (numRead == 0) {
            // Všechna data přečtena.
            return;
        }

        readBuffer.flip();

        // Předání dat ke zpracování.
        worker.process(socketChannel, readBuffer);
    }
}
```

Výše zmíněná ukázka obsahuje metodu `read()`, sloužící k načtení dat z aktuálního kanálu připraveného ke čtení. Z kódu je vidět, jak se přijatá data předávají ke zpracování

objektu `worker` přes metodu `process()`. V další kapitole bude ukázáno, jak se přijatá data následně zpracovávají.

Zápis dat probíhá v metodě `write()`, jejíž kód je k vidění v následující ukázce.

```
private void write(SelectionKey key) throws IOException {
    SocketChannel socketChannel = (SocketChannel)
        key.channel();

    synchronized (dataToWrite) {
        Queue<ByteBuffer> dataQueue =
            dataToWrite.get(socketChannel);

        Iterator<ByteBuffer> it = dataQueue.iterator();
        while (it.hasNext()) {
            ByteBuffer buf = (ByteBuffer) it.next();
            it.remove();
            socketChannel.write(buf);
            if (buf.remaining() > 0) {
                break;
            }
        }
        if (dataQueue.isEmpty()) {
            // Všechna data zapsána. Přepnutí na
            // naslouchání.
            key.interestOps(SelectionKey.OP_READ);
        }
    }
}
```

Proměnná `dataToWrite` představuje mapu, která uchovává pro daný kanál frontu dat k odeslání. Jelikož jsou tyto zprávy přidávány z jiného vlákna, konkrétně z vlákna `worker`, musí být přístup k této mapě synchronizován použitím sekce `synchronized`. Poté, co je získán přístup do této sekce, mohou se začít data z fronty odesílat. Když jsou všechna odeslána, nastaví se stav kanálu na režim čtení.

5.1.2 Zpracování zpráv

Zpracování zpráv obstarává samostatné vlákno, které je reprezentované třídou `IncomingWorker`. V něm jsou z dat opět vytvářeny příchozí zprávy, ke kterým je navíc přiřazován takzvaný kontext. Obojí je uloženo v instanci třídy `IncomingMessageServer`. Pro představu je uvedena i část kódu této třídy.

```

public class IncomingMessageServer {

    private MessageContext context;
    private MessageServer message;

    public IncomingMessageServer(MessageContext context,
        MessageServer message) {
        super();
        this.context = context;
        this.message = message;
    }

    :
}

```

Instance třídy *MessageContext* je tedy přítomna u každé zprávy přijaté z mobilního zařízení. Nese informace o aktuálním spojení a vzdáleném zařízení, které jsou relevantní pro zpracování dané zprávy. Zpracování zpráv totiž probíhá velmi podobným způsobem jako na aplikaci pro Android. Opět se u centrálního prvku pro manipulaci se zprávami registrují handlers (objekty pro zpracování příslušné zprávy), o jejichž použití rozhoduje podmínka filtru. Ten je zaregistrovaný společně s handlerem. V této architektuře by však bylo složité předávat informace o aktuální zpracovávané zprávě nějakým způsobem „zvenčí“. Proto jsou tyto informace dostupné u všech zpracovávaných zpráv. Jedná se zejména o identifikátor mobilního zařízení, jeho aktuální konfiguraci (počty a typy zařízení, senzorů) a kanál na kterém komunikuje pro případ, že by bylo potřeba poslat odpověď na zpracovávanou zprávu. Kontext je budován postupně s příchozími zprávami. Nejprve přijde zpráva s identifikátorem mobilního zařízení, což se zaznamená do kontextu. Poté se načte příslušná konfigurace z databáze, pokud je uložena. Pokud není, mobilní zařízení ji posílá v další zprávě, přičemž je uložena do databáze i do kontextu pro následující zprávy.

Atributy obsažené v tomto kontextu jsou k vidění v následující ukázce kódu.

```

public class MessageContext {

    private SocketChannel channel;
    private SocketAddress remoteAddress;

    private int androidId;
    private Android conf;

    private StatementsMap statementsMap;

    :
}

```

Za zmínku stojí i atribut *statementsMap*, který nese mapu objektů *PreparedStatement*, sloužících pro práci s databází. Jedná se o předkompilované příkazy jazyka SQL, zejména pro vkládání dat do databáze. Tyto příkazy jsou připraveny vždy, když se

zahájí zpracování dávky zpráv (příčemž je otevřeno i spojení s databází) a nesou i údaje o aktuálním spojení s databází. V samotném kódu handleru tedy není nutné řídit spojení s databází a jeho životní cyklus. Ten je řízen vláknem v třídě *IncomingWorker*. Jak vypadá smyčka zpracování příchozích zpráv je uvedeno níže.

```
try (Connection connection =
    DatabaseManager.getConnection();
    StatementsMap statementsMap = new
        StatementsMap(connection);) {

    connection.setAutoCommit(false);
    try {
        IncomingData incomingData = queue.take();

        Deque<IncomingMessageServer> messages =
            incomingData
                .getServerMessages();

        for (IncomingMessageServer message : messages) {
            try {
                message.getContext()
                    .setStatementsMap(statementsMap);
                dispatcher.handle(message);
            } catch (Exception e) {
                :
            }
        }

        sendAck(incomingData.channel);

        connection.commit();
    }

    :
```

Díky tomu, že třída *StatementsMap* implementuje rozhraní *AutoCloseable*, lze příslušný objekt použít v konstrukci zvané „try-with-resources“, která se postará o uvolnění jak spojení s databází, tak předkompilovaných SQL příkazů.

V kódu handleru je pak díky výše uvedenému postupu použití SQL příkazů *PreparedStatement* jednodušší, jak se lze přesvědčit v ukázce.

```

@Override
public void handle(IncomingMessageServer message) throws
    Exception {
        :
        PreparedStatement statementMeasurements =
            message.getContext()
                .getStatementsMap()
                .getPreparedStatement(MEASUREMENTS);
        PreparedStatement statementMeasurementValues =
            message.getContext()
                .getStatementsMap()
                .getPreparedStatement(MEASUREMENT_VALUES);

        // Vložení dat do databáze
        :
    }

```

5.2 Předvedení serverové aplikace

V této kapitole je představena serverová aplikace z hlediska použití. Jelikož se jedná o konzolovou aplikaci, která má za úkol běžet na serveru jako démon, bude pouze představeno spuštění z příkazové řádky. Celá aplikace poté již funguje sama bez intervence uživatele.

Připojení aplikace k databázi je konfigurovatelné skrze nastavení systémových parametrů. Toho je docíleno použitím parametru u příkazu `java` s přepínačem `-D` jako předponou názvu. Lze tedy například specifikovat systémový parametr `st32352.user` parametrem z příkazové řádky `-Dst32352.user="androidmeric"`. Při běhu aplikace bude mít parametr `st32352.user` hodnotu `androidmeric`.

Tabulka 3 – Výčet parametrů serverové aplikace.

typ parametru	název parametru	popis
systémový parametr	st32352.jdbc	JDBC řetězec pro připojení do PostgreSQL databáze.
	st32352.user	Uživatel v databázi.
	st32352.pwd	Heslo uživatele v databázi.
parametry metody <i>main()</i>	<code>-port (-p)</code>	Číslo portu, na kterém aplikace naslouchá pro spojení s mobilními zařízeními.

Jak je vidět z tabulky, lze také specifikovat port, na kterém bude aplikace naslouchat pro spojení s mobilními zařízeními. Jedná se však již o klasický parametr aplikace, který je předán ke zpracování metodě *main()*. Jak vypadá spuštění aplikace s databází na lokálním počítači a nasloucháním na portu 12345 lze shlédnout na následující ukázce.

```
$ java -Dst32352.jdbc="jdbc:postgresql://127.0.0.1:5432/androidmeric"  
-Dst32352.user="androidmeric" -Dst32352.pwd="mojeheslo12345"  
-jar "AndroidMericServer-0.0.1-SNAPSHOT-jar-with-dependencies.jar"  
--port 12345
```

```
Kvě 05, 2015 10:40:35 ODP. st32352.androidmeric.server.
```

```
AndroidMericServer main
```

```
INFO: Starting application
```

```
Kvě 05, 2015 10:40:35 ODP. st32352.androidmeric.server.comm.
```

```
AndroidCommunicator run
```

```
FINER: ENTRY
```

```
Kvě 05, 2015 10:40:35 ODP. st32352.androidmeric.server.comm.
```

```
AndroidCommunicator init
```

```
INFO: Listening on port 12345
```


6 Webová aplikace

Poslední aplikací, kterou ještě zbývá představit, je webová aplikace pro nasazení na serveru. Právě jejím popisem se tato kapitola zabývá. Postupně bude uveden účel, za jakým tato aplikace vznikla, její návrh, implementační detaily a nakonec i ukázka z hlediska uživatele.

Webová aplikace slouží uživatelům k zobrazení naměřených hodnot z jednotlivých mobilních zařízení a také ke zobrazení jejich konfigurace. Tomu také odpovídá rozložení aplikace, která je rozdělena na dvě příslušné části.

Pro vývoj byl použit javovský framework Vaadin, který zprostředkovává prezentační vrstvu aplikace. Obsahuje mnoho předpřipravených grafických komponent, které jsou ke shlédnutí na oficiálních stránkách tohoto projektu¹. Tato volba ovlivnila i architekturu aplikace, jelikož je silně postavena na technologii AJAX. Grafické rozhraní je tak tvořeno jednou hlavní obrazovkou, ve které jsou komponenty vyměňovány dle potřeby. Výsledná aplikace tak má velice blízko ke klasickým desktopovým aplikacím.

Za základní stavební komponentu grafického rozhraní lze považovat rozhraní *View*, které představuje jednu obrazovku aplikace. Mezi instancemi tříd, které toto rozhraní implementují, lze navigovat pomocí třídy *Navigator*. U ní si lze jednotlivé instance *View* zaregistrovat pro konkrétní části URL adresy. Pokud je název aplikace **AndroidMericWeb**, pak základní adresa aplikace má tvar `http://localhost:8080/AndroidMericWeb/`. Při registraci například s řetězcem `conf` je výsledná adresa `http://localhost:8080/AndroidMericWeb/#!conf`. V aplikaci jsou dvě třídy implementující výše uvedené rozhraní — jedna pro práci s naměřenými hodnotami a druhá pro zobrazení konfigurací.

Ve webové aplikaci je použit i framework Spring. Ten zejména spravuje dependency injection, datové zdroje aplikace a transakce při práci s JPA. Vaadin totiž sám o sobě neumožňuje dependency injection z toho důvodu, že používá servlet (kde je EJB dependency injection možné) pro své vnitřní účely a uživatele od této funkčnosti izoluje. Tím však také umožňuje vytváření grafického rozhraní prostřednictvím programování kódu v jazyce Java. Například ve frameworku JSF jsou jednotlivé stránky reprezentovány XML kódem a jsou spravovány webovým kontejnerem. Ten umožňuje práci s takzvanými managed beanami, které jsou poté „vstříknuty“ na své místo ve stránce. Jelikož však třídy, které reprezentují jednotlivé komponenty ve Vaadinu, takto spravovány nejsou, nelze již přítomné dependency injection použít (ať už EJB nebo managed beany, které jsou navíc specifické pro framework JSF). Lze však použít framework Spring, protože Vaadin umožňuje nastavit třídy (konkrétně instance rozhraní *View*) pro správu přes kontejner Springu za použití anotací. Další použité technologie jsou popsány podrobněji v kapitole pojednávající o použitých technologiích a podkapitole Webová aplikace.

¹Stránka s ukázkami komponent — <http://demo.vaadin.com/sampler/>

6.1 Návrh a implementace webové aplikace

Tato kapitola se zabývá popisem jednotlivých částí webové aplikace. Jak již bylo řečeno, existují hlavní dvě části aplikace — jedna pro zobrazení naměřených hodnot a druhá pro zobrazení konfigurací jednotlivých mobilních zařízení. Obě tyto části budou představeny v následujících kapitolách.

6.1.1 Zobrazení naměřených hodnot — `DefaultView`

Pro zobrazení naměřených hodnot je použita třída `DefaultView`, která, jak už název napovídá, implementuje rozhraní `View`. Díky tomu lze na tuto stránku navigovat přes URL adresu, která je v tomto případě v základním tvaru. Ten byl zmíněn již v úvodu k této aplikaci a má tvar `http://localhost:8080/AndroidMericWeb/`. Pro to, aby byly instance této třídy spravovány kontejnerem Springu, bylo třeba označit deklaraci třídy příslušnou anotací. Poté již lze používat anotaci `@Autowired` u atributů třídy, kde se má provést dependency injection.

```
@SpringView(name = DefaultView.VIEW_NAME)
public class DefaultView extends HorizontalLayout
    implements View {
    public static final String VIEW_NAME = "";

    @Autowired
    private AndroidManager am;

    @Autowired
    private MeasurementManagerBean measurementManager;

    private DeviceManager dm;
    :
}
```

Ve výše uvedené ukázce je možné vidět tři atributy, které plní na dané stránce nejdůležitější úkony. Proto budou probrány podrobněji.

Prvním atributem je `am` typu `AndroidManager`. Ten slouží k načítání konfigurace pro příslušná mobilní zařízení přes JPA. Obsahuje metodu `getAndroidById()`, která jako parametr přijímá identifikační číslo mobilního zařízení a navrácí načtenou konfiguraci z databáze. V případě této webové stránky však slouží pouze jako zprostředkovatel instance `EntityManager`, která následně slouží Vaadinu pro načítání konfigurace do tabulek. Kód této krátké třídy je uveden v následujícím příkladu. Za povšimnutí stojí použití anotace Vaadinu `@SpringComponent`, která označuje třídy jako komponenty frameworku Spring. Tím je umožněna správa životního cyklu instancí této třídy springovským kontejnerem. Vaadin však používá odlišné životní cykly spravovaných komponent než jaké Spring běžně nabízí. Je proto potřeba používat Vaadinem zprostředkované anotace, jako je například v následující ukázce `@UIScope`. K tomuto tématu je však

zatím poměrně málo dostupných informací, jelikož integrace Springu a Vaadinu je stále poměrně novou záležitostí (i během psaní tohoto textu je projekt stále ve verzi beta). Většina informací byla čerpána z nového oficiálního návodu na stránkách projektu Vaadin (SARA, 2015).

```
@SpringComponent
@UIScope
public class AndroidManager {

    @PersistenceContext(unitName = "DeviceConfig")
    private EntityManager entityManager;

    public Android getAndroidById(int androidId) {
        if (entityManager != null) {
            TypedQuery<Android> androidQuery =
                entityManager.createNamedQuery(
                    "Android.findById", Android.class);
            androidQuery.setParameter("id", androidId);
            return androidQuery.getSingleResult();
        }
        return null;
    }

    public EntityManager getEntityManager() {
        return entityManager;
    }
}
```

Dalším důležitým atributem třídy *DefaultView* je atribut `measurementManager` typu *MeasurementManagerBean*. Je to instance třídy, která má na starost načítání naměřených hodnot z databáze. Jedná se opět o komponentu Springu, která je „vstříknuta“ do stránky. Obsahuje dvě metody, jednu pro načtení všech hodnot z mobilního zařízení s požadovaným identifikátorem a v zadaném časovém rozsahu a druhou pro načtení posledních uložených hodnot ze zařízení s daným identifikátorem.

Posledním atributem třídy *DefaultView*, který je potřeba zmínit, je atribut `dm` typu *DeviceManager*. Jedná se o třídu, která slouží ke zobrazení komponenty s měřicími zařízeními — té samé, jenž je použita v mobilní aplikaci. Tím je docíleno jednotného vzhledu v obou aplikacích. Tato třída rozšiřuje abstraktní třídu *AbstractJavaScriptComponent* z frameworku Vaadin, sloužící k reprezentaci komponent, jejichž celá logika je na klientské straně ve formě JavaScriptu. Instance této třídy na serveru komunikuje s klientskou stranou skrze stav, který se získá voláním metody *getState()*. Ten nese data komponenty, s nimiž lze pracovat na klientské straně. Tento stav může nést základní datové typy jazyka Java, které jsou přeloženy na straně JavaScriptu do příslušných proměnných. Třída se stavem komponenty s měřicími zařízeními vypadá následovně.

```

public class DeviceManagerState extends
    JavaScriptComponentState {

    private String devicesString;
    private String values;

    // Gettery a settery
                                :
}

```

V atributu `devicesString` jsou obsaženy údaje o konfiguraci měřicího zařízení a v atributu `values` zase o aktuálních hodnotách. Oba tyto atributy nesou údaje ve formátu JSON pro snadné zpracování na klientské straně. V JavaScriptovém kódu je také potřeba definovat funkci `onStateChange()`, která je volána při změně hodnot atributů. Jak vypadá hlavní část kódu klientské části je možné vidět v příloze G

6.1.2 Zobrazení konfigurací — *AndroidsView*

V této kapitole je věnována pozornost především stránce pro zobrazení konfigurací mobilních zařízení. Přesněji konfigurací měřicích zařízení, které jsou k mobilním zařízením připojeny. Tuto stránku reprezentuje třída *AndroidsView*, která je opět implementací rozhraní *View*. Tentokrát je u navigátoru registrována s názvem `conf` a výsledná URL má tedy tvar `http://localhost:8080/AndroidMericWeb/#!conf`. Pro anotace za účelem použití kontejneru Springu pro správu této komponenty platí to samé, co u předchozí třídy pro zobrazení naměřených hodnot.

```

@SpringView(name = AndroidsView.VIEW_NAME)
public class AndroidsView extends HorizontalLayout
    implements View,
        DeviceServerSelectionListener {
    public static final String VIEW_NAME = "conf";

    @Autowired
    private AndroidManager am;

    private JPAContainer<Android> androidsDB;
    private Table androidsTable;

    private AndroidDetail androidDetail = new
        AndroidDetail();
    private DeviceDetail deviceDetail = new DeviceDetail();
                                :
}

```

V této třídě je opět využito atributu `am` typu *AndroidManager*, avšak, stejně jako v předchozím případě, pouze jako zdroje instance *EntityManager* pro práci s JPA.

Jednotlivá mobilní zařízení jsou zobrazována v tabulce, kde je možné požadované zařízení vybrat. Poté se zobrazí příslušné detaily. Jejich zobrazení uživateli obstarávají další dvě komponenty představované atributy `androidDetail` a `deviceDetail`. Jak už názvy těchto komponent napovídají, slouží ke zobrazení detailních informací o mobilním zařízení, respektive o vybraném měřicím zařízení. Více informací s ukázkou bude uvedeno v následující kapitole věnující se představení aplikace z hlediska uživatele.

6.2 Předvedení webové aplikace

Tato kapitola se zabývá popisem webové aplikace z uživatelského hlediska. Aplikace umožňuje uživateli zobrazovat mobilní zařízení uložená v databázi a jejich konfigurace. Zejména pak příslušná měřicí zařízení a senzory a informace s těmito objekty spojené. Také přehledně prezentuje naměřené hodnoty z těchto zařízení.

Hlavní obrazovka aplikace obsahuje přehled uložených mobilních zařízení a jejich aktuálních hodnot. Na tuto obrazovku se lze dostat přes základní adresu aplikace, tedy URL ve tvaru `http://localhost:8080/AndroidMericWeb/`, jak již bylo zmíněno, nebo stisknutím tlačítka na horní liště s nápisem „Hlavní“. Jak tato obrazovka vypadá je k vidění na obrázku 17. Na levé straně obrazovky je tabulka s uloženými mobilními zařízeními. Po výběru libovolného zařízení se v pravé části obrazovky ukáže přehled s aktuálními hodnotami, již známý z aplikace pro mobilní zařízení.

Hlavní	Zařízení Android	
--------	------------------	--

Android zařízení

ID
3
4

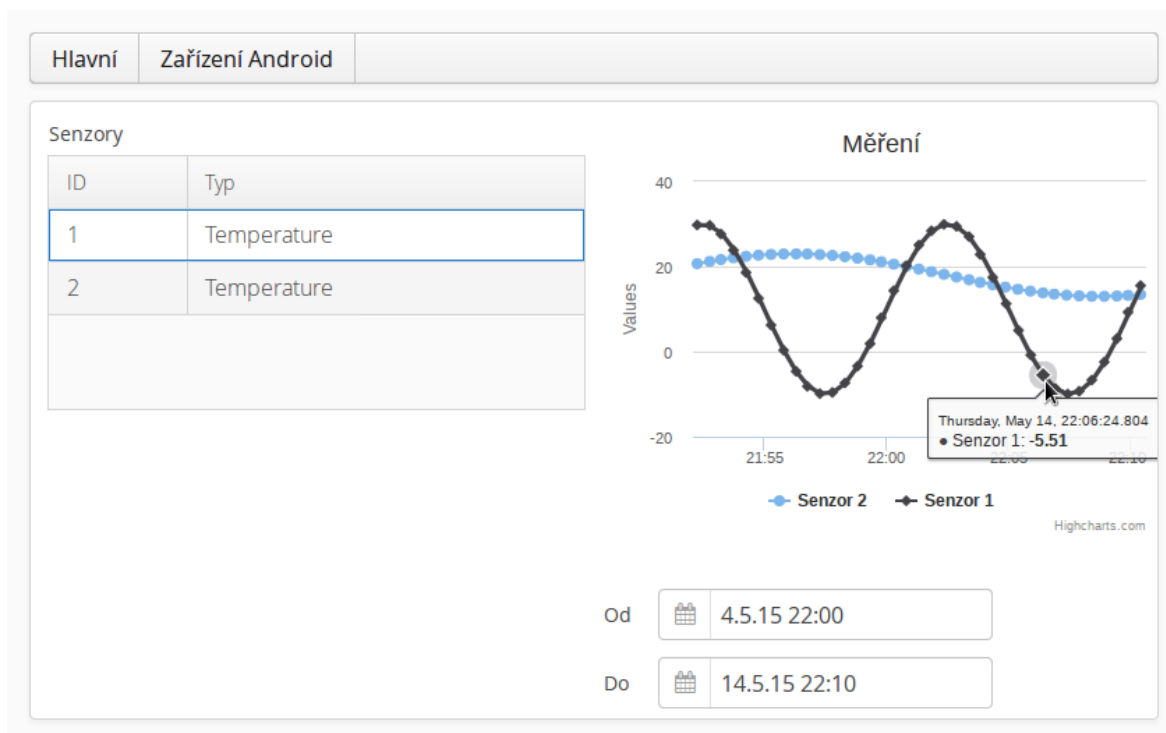
Měřicí zařízení 0	
Senzor 1	Senzor 2
29.20°C	22.89°C

Měřicí zařízení 1	
Senzor 0	Senzor 1
20.62°C	0

Obrázek 17 – Obrazovka webové aplikace s aktuálními hodnotami z mobilních zařízení.

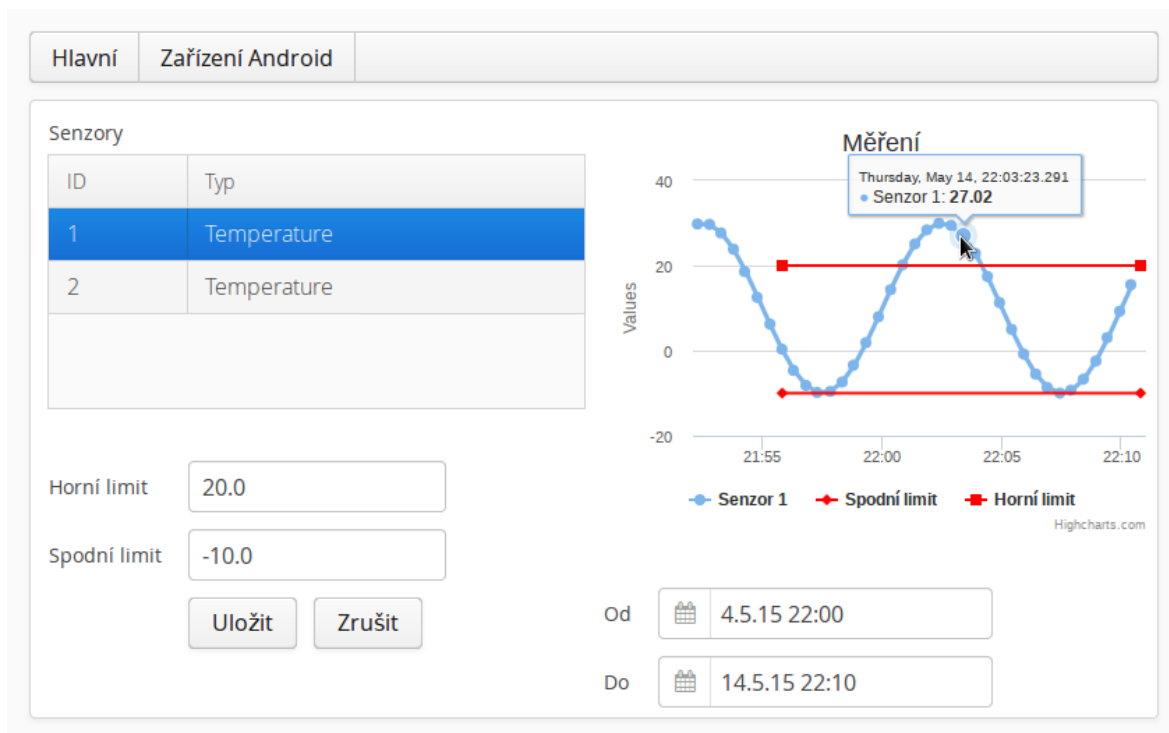
Po stisknutí grafického prvku měřicího zařízení je, stejně jako v mobilní aplikaci, zobrazen graf naměřených hodnot. V levé části obrazovky je tabulka se senzory přísluš-

ného měřicího zařízení, v pravé části je graf s naměřenými hodnotami ze všech senzorů. Tento graf lze přibližovat stisknutím a horizontálním tažením levého tlačítka myši nad grafem. Tím dojde k výběru požadovaného časového rozsahu. V pravé spodní části obrazovky jsou k dispozici dvě vstupní pole pro zadání od jakého data do kterého se mají uložené hodnoty načíst. Obrazovka s grafem všech hodnot z měřicího zařízení je na obrázku 18.



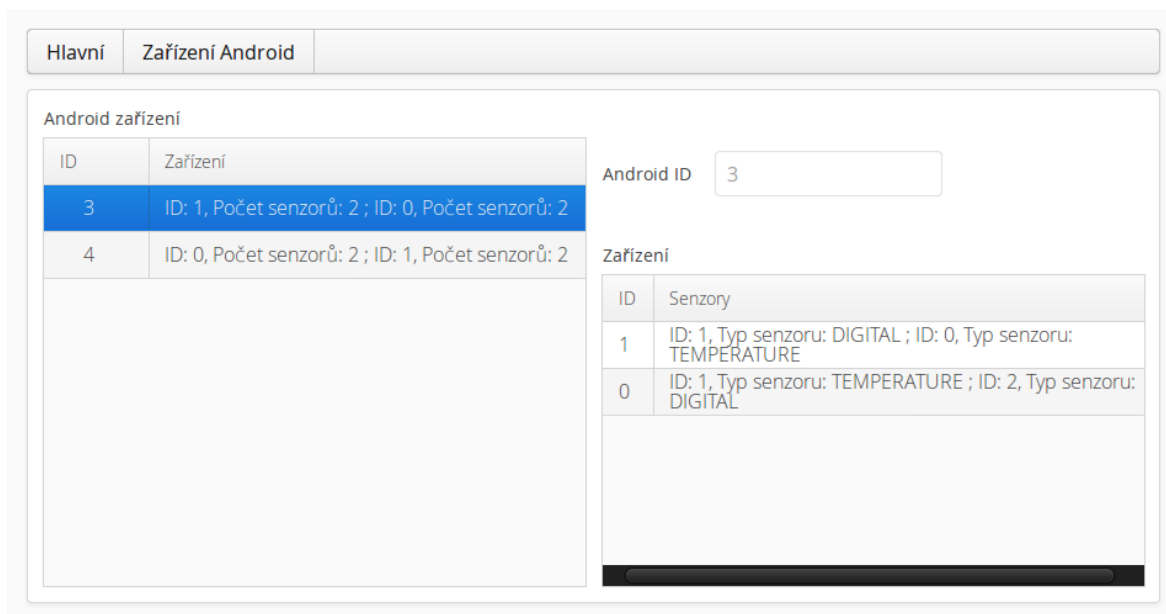
Obrázek 18 – Obrazovka webové aplikace s grafem všech naměřených hodnot.

Pokud je v levé tabulce vybrán konkrétní senzor, dojde ke změně obrazovky. V grafu se vykreslí pouze naměřené hodnoty pro daný senzor a k tomu se vykreslí i případné limitní hodnoty. Ty lze upravovat změnou údajů ve vstupních polích, které se po výběru senzoru objevily pod tabulkou senzorů. Pokud již byly nějaké limitní hodnoty zadány, jsou v těchto vstupních polích vyplněny. Pokud jsou zadány nové hodnoty, v grafu dojde k jejich vykreslení až od času jejich uložení do systému. Jsou tedy zobrazeny i historické limitní hodnoty v příslušných časových úsecích jejich platnosti. Jak vypadá obrazovka s jedním vybraným senzorem je zobrazeno na obrázku 19.



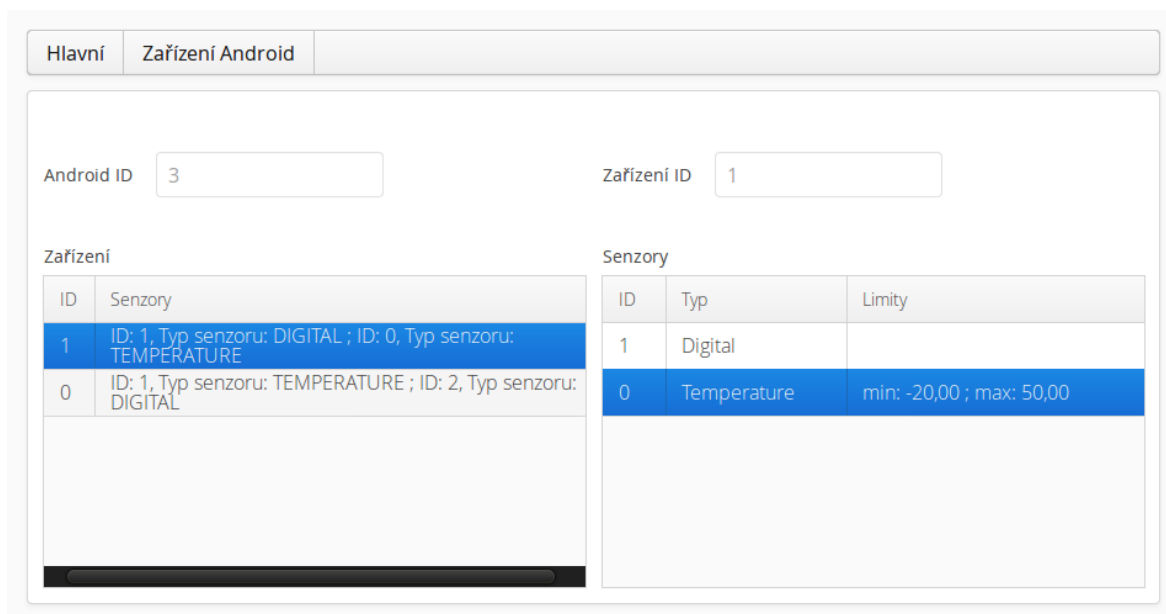
Obrázek 19 – Obrazovka webové aplikace s grafem naměřených hodnot ze senzoru.

Druhou hlavní obrazovkou je obrazovka s konfigurací mobilních zařízení. Základní URL je ve tvaru <http://localhost:8080/AndroidMericWeb/#!conf>. Na tuto adresu je uživatel odkázán po stisknutí tlačítka „Zařízení Android“. V levé části obrazovky je tabulka s uloženými mobilními zařízeními. Po výběru zařízení se v pravé části obrazovky zobrazí podrobnější informace ohledně mobilního zařízení, zejména ID a další tabulka s příslušnými měřicími zařízeními. Zde si lze vybrat libovolné zařízení pro zobrazení detailu, což je vidět na obrázku 20.



Obrázek 20 – Obrazovka webové aplikace s konfiguracemi mobilních zařízení.

Po výběru požadovaného měřicího zařízení se obrazovka změní. Tentokrát se podrobnosti o mobilním zařízení společně s tabulkou s měřicími zařízeními zobrazí v levé části obrazovky. Pravou část obrazovky zabere zobrazení detailních informací o vybraném měřicím zařízení společně s tabulkou příslušných senzorů. Tomuto odpovídá obrázek 21.



Obrázek 21 – Obrazovka webové aplikace s konfiguracemi měřicích zařízení.

Závěr

Cílem diplomové práce bylo vytvoření systému aplikací pro měření hodnot na mobilních zařízeních a jejich následné zpracování na serveru. Nejprve byl navržen systém senzorů, měřicích a komunikačních zařízení, které mají za úkol měřit hodnoty a předávat je do mobilní aplikace. V praxi by se jednalo o hardwarová zařízení. Jelikož však cílem práce bylo pouze vytvoření systému aplikací, byla tato zařízení simulována. K tomuto účelu byl použit jazyk Lua, jehož interpret jejich činnost simuloval. To umožnilo tvorbu dynamických konfiguračních souborů, ve skutečnosti skriptů jazyka Lua. Bylo tedy například možné vytvořit v konfiguračním souboru funkci, která pro daný senzor generuje hodnoty na základě vstupního parametru času. Aplikace pro mobilní zařízení je však navržena tak, aby bylo možné jednoduše vyměnit simulované zařízení za reálné. Stačí vytvořit třídy s požadovanou funkčností a s příslušným rozhraním.

První část diplomové práce představila architekturu a návrh systému měření hodnot, účel jednotlivých aplikací a technologie v těchto aplikacích použité. V druhé části byly popsány implementační detaily a jednotlivé vytvořené aplikace. Z důvodu zachování přehlednosti písemné části práce byly u každé aplikace popsány pouze nejdůležitější části.

V praktické části diplomové práce byl vytvořen systém tří aplikací. První aplikací je mobilní aplikace, která má za úkol získávání naměřených hodnot, jejich uchování a jednoduché zobrazení a také odesílání na server. Druhá aplikace je určená k nasazení na server a slouží ke komunikaci s mobilními zařízeními. Přijímá zprávy z těchto zařízení, zpracovává je a ukládá příslušná data do databáze. Třetí a poslední aplikací je webová aplikace. Ta zobrazuje data uložená v databázi. Dokáže tedy například zobrazovat uložené hodnoty ve formě grafu nebo uložené konfigurace mobilních zařízení. Vzhledem k počtu použitých technologií se jednalo o nejkomplexnější aplikaci. Bylo poměrně obtížné nakonfigurovat jednotlivé technologie pro vzájemnou spolupráci. Například lze zmínit případ, kdy knihovna EclipseLink nespolečně pracovala správně s externě řízenými JTA transakcemi. Tento problém byl však odstraněn.

V praktickém nasazení by bylo vhodné upravit formát zpráv mezi komunikačním a mobilním zařízením — přidat některá nová pole, například délku zprávy. Ta byla z komunikace se simulovaným zařízením vypuštěna, jelikož, jak je zmíněno v příslušné kapitole, nebyla potřeba. Rovněž by pravděpodobně bylo potřeba přidat více atributů do konfigurace, například s popisem jednotlivých zařízení nebo i senzorů pro jejich jednodušší identifikaci. Jednoduchým způsobem by také bylo možné v případě potřeby přidat volbu komunikace mobilního zařízení se serverem přes mobilní síť. Systém Android je schopný sám rozhodnout zda pro přístup na internet použít Wi-Fi nebo mobilní síť. Tuto možnost je však nutné nejdříve aktivovat.

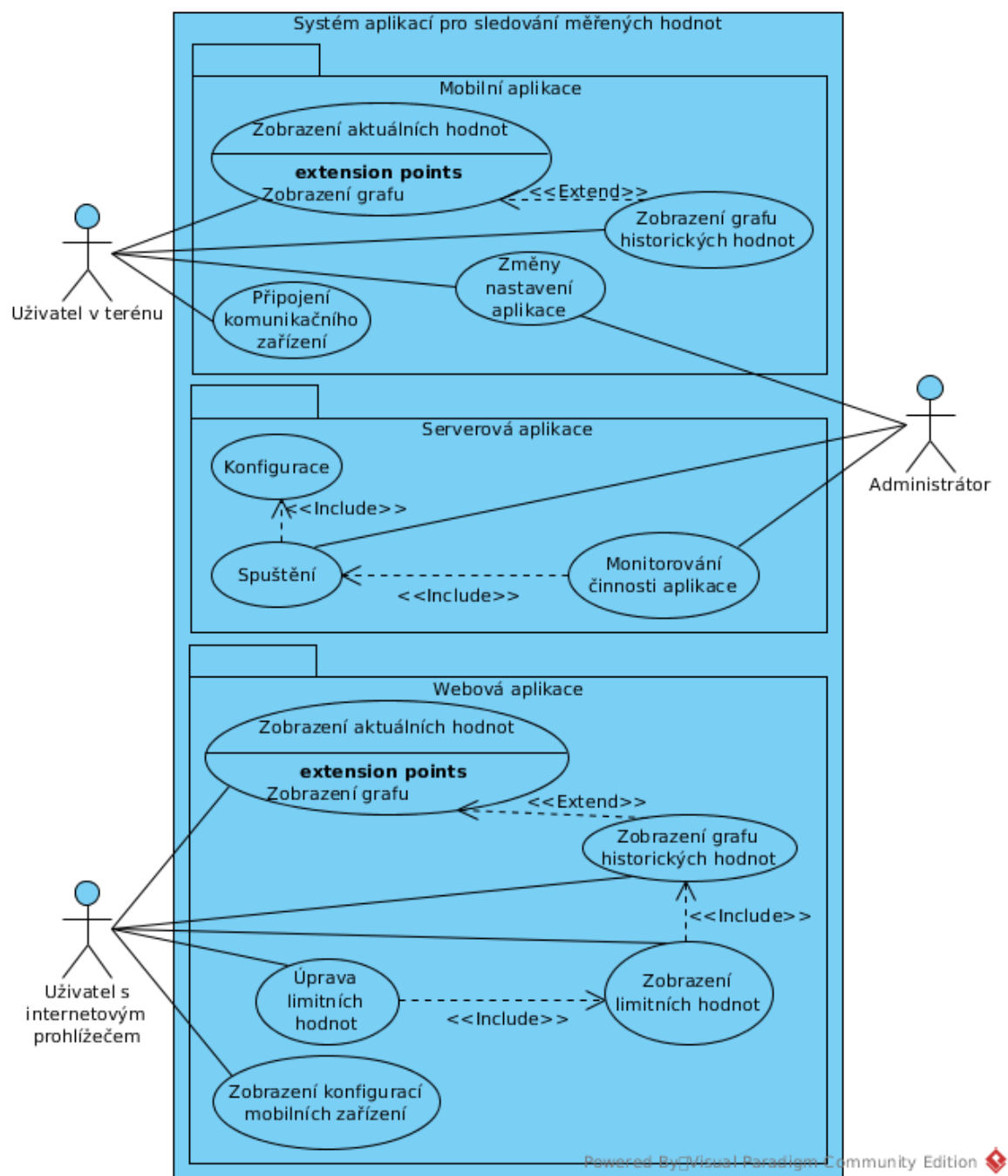
Literatura

- ComponentJS*. [online]. 2009 [cit. 2015-04-19]. Dostupný z WWW: <http://componentjs.com>.
- DEBIAN DOCUMENTATION TEAM. 2013. *A Brief History of Debian: Chapter 3 - Debian Releases* [online]. 2013 [cit. 2015-04-20]. Dostupný z WWW: <https://www.debian.org/doc/manuals/project-history/ch-releases.en.html>.
- DUDLEY, Brier. 2014. *Microsoft reveals Windows 10, with hybrid Start menu* [online]. 2014 [cit. 2015-05-09]. Dostupný z WWW: <http://blogs.seattletimes.com/brierdudley/2014/09/29/microsoft-previews-windows-9/>.
- ENGLUND, Marc. 2012. *Integrating an existing GWT widget* [online]. 2012 [cit. 2015-04-23]. Dostupný z WWW: <https://vaadin.com/wiki/-/wiki/Main/Integrating%20an%20existing%20GWT%20widget>.
- FERREIRA, Vasco. 2014. *How does Spring @Transactional Really Work?* [online]. 2014 [cit. 2015-05-01]. Dostupný z WWW: <http://java.dzone.com/articles/how-does-spring-transactional>.
- GEMIUS SA. 2000. *Operating systems* [online]. 2000 [cit. 2015-05-08]. Dostupný z WWW: <http://www.rankings.cz/en/rankings/operating-systems.html>.
- GOOGLE. 2015a. *AlarmManager* [online]. 2015 [cit. 2015-05-01]. Dostupný z WWW: <http://developer.android.com/reference/android/app/AlarmManager.html>.
- GOOGLE. 2015b. *WifiManager.WifiLock* [online]. 2015 [cit. 2015-05-01]. Dostupný z WWW: <http://developer.android.com/reference/android/net/wifi/WifiManager.WifiLock.html>.
- HARDY, Brian. 2013. *Android programming: the big nerd ranch guide*. 1st ed. Atlanta, GA: Big Nerd Ranch, 2013. 580 s. ISBN 03-218-0433-3.
- HIGHCHARTS. 2015a. *About Us* [online]. 2015 [cit. 2015-04-24]. Dostupný z WWW: <http://www.highcharts.com/about>.
- HIGHCHARTS. 2015b. *Highcharts* [online]. 2015 [cit. 2015-04-24]. Dostupný z WWW: <http://www.highcharts.com/products/highcharts>.
- IERUSALIMSCHY, Roberto. 2013. *Programming in Lua*. 3rd ed. Rio de Janeiro: Lua.org, 2013. 348 s. ISBN 978-8590379850.
- MEIER, Reto. 2012. *Professional Android 4 application development*. Updated for Android 4. Indianapolis: John Wiley, 2012. 817 s. ISBN 978-111-8262-153.
- MVNREPOSITORY. 2014. *Luaj Jse* [online]. 2014 [cit. 2015-04-20]. Dostupný z WWW: <http://mvnrepository.com/artifact/org.lua-jse/luaj-jse/3.0>.
- NET APPLICATIONS.COM. 2006. *Mobile/Tablet Operating System Market Share* [online]. 2006 [cit. 2015-04-20]. Dostupný z WWW: <http://marketshare.hitslink.com/operating-system-market-share.aspx?qprid=8&qpcustomid=1&qpsp=183&qpn=13&qptimeframe=M>.

- ORACLE. 2007. *Eclipse Persistence Platform (EclipseLink) FAQ* [online]. 2007 [cit. 2015-04-19]. Dostupný z WWW: <http://web.archive.org/web/20070311212527/http://www.oracle.com/technology/tech/eclipse/pdf/eclipselink-faq.pdf>.
- ORACLE CORPORATION AND/OR ITS AFFILIATES. 2014. *JSRs: Java Specification Requests: JSR 51: New I/O APIs for the Java™ Platform* [online]. 2014 [cit. 2015-04-23]. Dostupný z WWW: <https://www.jcp.org/en/jsr/detail?id=51>.
- PANDA, Debu; RAHMAN, Reza. 2014. *EJB 3 in action*. Second edition. Shelter Island: Manning, 2014. 560 s. ISBN 978-193-5182-993.
- RED HAT, INC. 2006. *Red Hat Completes Acquisition of JBoss: Open source leader enables a low-cost path to service-oriented architectures* [online]. 2006 [cit. 2015-04-23]. Dostupný z WWW: <http://www.redhat.com/en/about/press-releases/jboss-final>.
- SARA, Henri. 2015. *Vaadin Spring* [online]. 2015 [cit. 2015-05-07]. Dostupný z WWW: <https://vaadin.com/wiki/-/wiki/Main/Vaadin+Spring>.
- SATERNOS, Casimir. 2014. *Client-server Web Apps With Javascript and Java*. 1st edition. Sebastopol, CA: O'Reilly Media, 2014. 260 s. ISBN 978-144-9369-330.
- SQLite Download Page*. [online]. [cit. 2015-04-20]. Dostupný z WWW: <http://sqlite.org/download.html>.
- THE ECLIPSE FOUNDATION. 2008. *Eclipse Announces EclipseLink Project to Deliver JPA 2.0 Reference Implementation* [online]. 2008 [cit. 2015-04-20]. Dostupný z WWW: https://eclipse.org/org/press-release/20080317_Eclipselink.php.
- THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. 1996a. *License* [online]. 1996 [cit. 2015-04-19]. Dostupný z WWW: <http://www.postgresql.org/about/licence>.
- THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. 1996b. *Porting from Oracle PL/SQL* [online]. 1996 [cit. 2015-04-19]. Dostupný z WWW: <http://www.postgresql.org/docs/9.1/static/plpgsql-porting.html>.
- THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. 2008. *Happy Birthday, PostgreSQL* [online]. 2008 [cit. 2015-04-19]. Dostupný z WWW: <http://www.postgresql.org/about/news/978/>.
- VAADIN LTD. 2015. *Pro Tools* [online]. 2015 [cit. 2015-04-23]. Dostupný z WWW: <https://vaadin.com/pro-tools>.
- W3TECHS. 2009. *Usage statistics and market share of Linux for websites* [online]. 2009 [cit. 2015-04-20]. Dostupný z WWW: <http://w3techs.com/technologies/details/os-linux/all/all>.
- WATSON, Gray. *OrmLite Releases* [online]. [cit. 2015-04-20]. Dostupný z WWW: <http://ormlite.com/releases/>.
- WIGLEY, Andy. 2014. *Universal Apps: What are they and how are they good for developers?* [online]. 2014 [cit. 2015-05-09]. Dostupný z WWW: <http://www.microsoft.com/en-gb/developers/articles/week03jul14/universal-apps-what-are-they-and-how-are-they-good-for-developers>.

WIKIPEDIA. 2015. *Spring Framework* — *Wikipedia, The Free Encyclopedia* [online]. 2015 [cit. 2015-04-24]. Dostupný z WWW: http://en.wikipedia.org/w/index.php?title=Spring_Framework&oldid=658796826.

Příloha A Use case diagram systému aplikací



Příloha B sensorView.js controller

```
// Komponenta controller
am.SensorView = cs.clazz({
  mixin: [ cs.marker.controller ],
  // Definice atributů třídy
  dynamics : {
    data : null
  },
  // Definice konstrukturu
  cons : function(sensorData) {
    this.data = sensorData;
  },
  // hash název/funkce třídy
  protos: {
    create: function () {
      // vytvoření modelu a view
      cs(this).create(
        "sv-model/sv-view",
        new am.SensorView.model(this.data),
        am.SensorView.view
      );

      this.data = null;
    }
  }
})
```

Příloha C sensorView.js model

```
// Komponenta model
am.SensorView.model = cs.clazz({
  mixin: [ cs.marker.model ],
  // Definice atributů třídy
  dynamics : {
    id :          null,
    valueType:    null
  },
  // Definice konstrukturu
  cons : function(sensorData) {
    var data = sensorData;
    this.id = data.id;
    this.valueType = data.valueType;
  },
  protos: {
    create: function () {
      /* Vytvoření modelu s atributy */
      cs(this).model({
        "data:id":      { value: this.id, valid:
                          "number" },
        "data:value":   { value: "N/A", valid:
                          "string" },
        "data:valueType": { value:
                          this.valueType, valid: "number" }
      })
    }
  }
})
```

Příloha D sensorView.js view

```
// Komponenta view
am.SensorView.view = cs.clazz({
  mixin: [ cs.marker.view ],
  protos: {
    render: function () {
      var self = this;
      /* Materializování UI struktury */
      var ui = $.parseHTML('<div
        class="sensorView">' +
          '<div class="id"></div>' +
          '<div class="value"></div>' +
          '</div>');

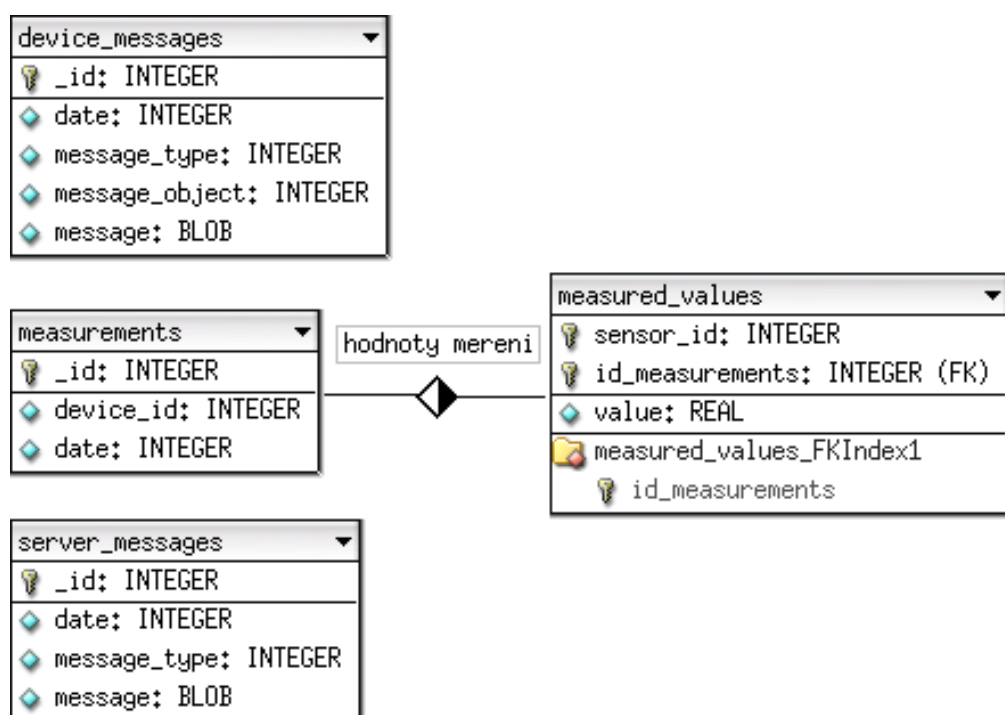
      /* Přidání do DOM rodičovské komponenty */
      cs(this).plug({ object: ui, spool:
        "materialized" });

      /* Naslouchání na změnu atributu data:id
        modelu */
      cs(this).observe({
        name: "data:id",
        spool: "materialized",
        touch: true,
        func: function (ev, label) {
          $(ui).find(".id").html("Senzor "+label)
        }
      });

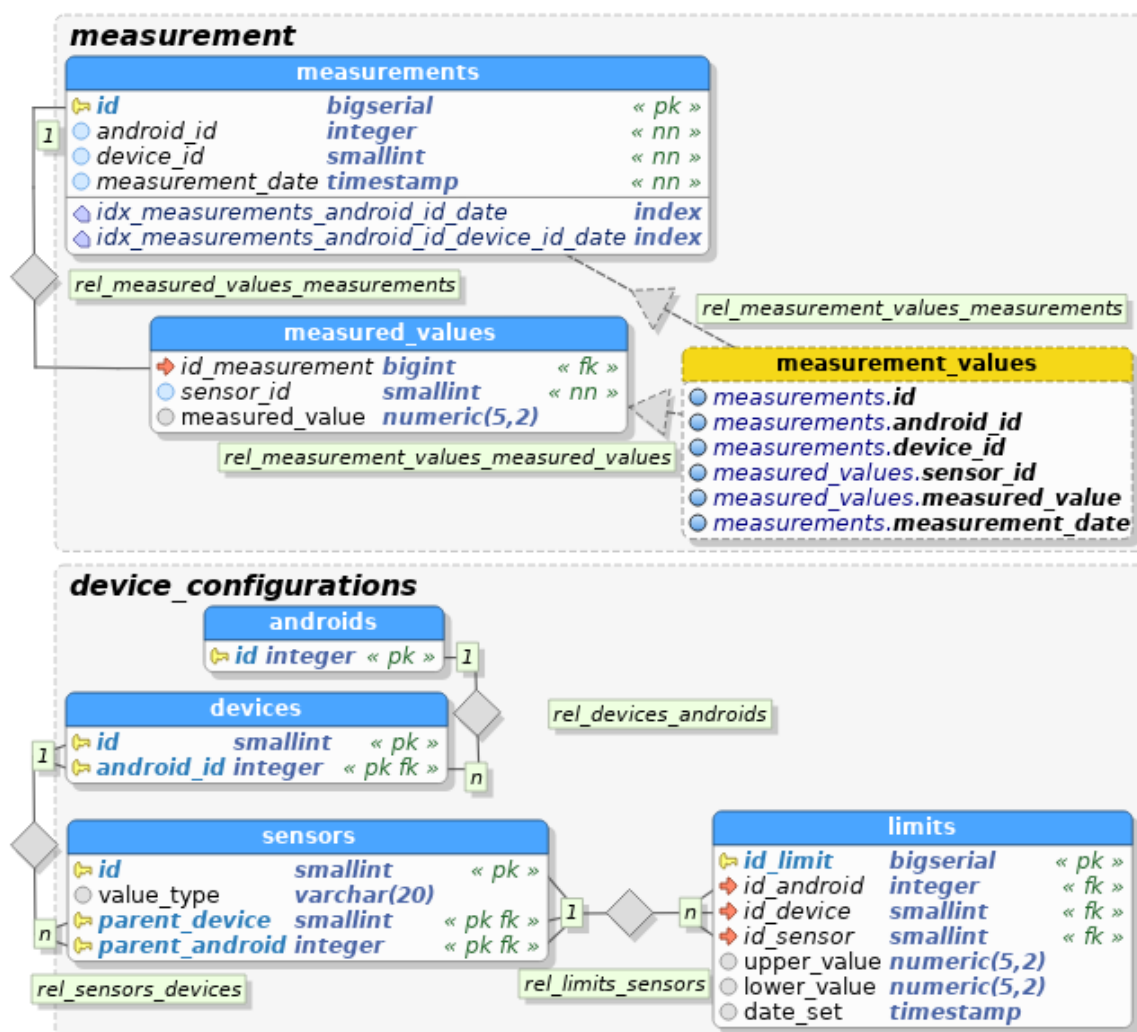
      :

      }
      $(ui).find(".value").html(html);
    }
  });
},
release: function () {
  /* Odpojení z DOM modelu, odregistrování
    naslouchání změn modelu */
```


Příloha E Model databáze měřicího zařízení



Příloha F Model databáze serveru



Příloha G Kód JavaScriptové komponenty webové aplikace

```
window.st32352_androidmeric_web_devicesView_DeviceManager
= function() {
    :
    window.deviceViewHandler = {}
    window.deviceViewHandler.onDeviceViewClick =
        function(deviceID) {
            var date = new Date();
            var id = deviceID.toString();
            log(date + ": dv" + id + ": clicked");

            var str = window.location.href;
            var lastSlash = str.lastIndexOf("/");
            var afterLastSlash = str.substring(lastSlash + 1,
                str.length);
            if (typeof parseInt(afterLastSlash) == "number") {
                window.location.href = window.location.href +
                    "/" + id;
                destroyDeviceManager();
            }
        }
    window.deviceViewHandler.onDeviceViewLongClick =
        function(deviceID) {
            var date = new Date();
            var id = deviceID.toString();
            log(date + ": dv" + id + ": longclicked");
        }

    this.onStateChange = function() {
        log("State change");
        var devicesData = this.getState().devicesString;
        log("DevicesData=" + devicesData);
        var values = this.getState().values;
        log("values=" + values);
        if (typeof devicesData != "undefined") {
            main(devicesData);
        }
        if (typeof values != "undefined") {
            setValues(values);
        }
    }
}
```