České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Tomáš Frček**

Studijní program: Otevřená informatika (magisterský)
Obor: Počítačové inženýrství

Název tématu: **Návrh aplikace pro měření znečištění ovzduší - projekt Kanárci**

Pokyny pro vypracování:

1. Seznamte se s problematikou měření znečištění ovzduší.
2. Navrhněte hardwarový přípravek pro měření koncetrace prachových částic včetně komunikačního modulu.
3 Navrhněte aplikaci pro OS Android, která bude vizualizovat údaje o znečištění.
4. Otestujte funkcionalitu přípravku v běžném provozo po dobu min. 1 týdne.

Seznam odborné literatury:

[1] P. Dutta, Common Sense: participatory urban sensing using a network of handheld air quality monitors, Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, Pages 349-350, 200
[2] A.R. Al-AliA Mobile GPRS-Sensors Array for Air Pollution Monitoring, vol. 10(10), Sensors Journal, 2010

Vedoucí: Ing. Daniel Novák, Ph.D.

Platnost zadání: do konce letního semestru 2014/2015

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 14. 1. 2014

České vysoké učení technické v Praze

Fakulta elektrotechnická

Katedra řídící techniky

Diplomová práce

Návrh aplikace pro měření znečištění ovzduší – projekt Kanárci

Bc. Tomáš Frček

Vedoucí práce: Ing. Daniel Novák, Ph.D.

Studijní program: Otevřená informatika

Obor: Počítačové inženýrství

5. května 2014

Poděkování

Děkuji vedoucímu diplomové práce Ing. Danielu Novákovi, PhD za odborné vedení a organizaci.

Děkuji Ing. Jakubu Hyblerovi za odborné rady, konzultace a pomoc při zpracování této práce.

Děkuji Bc. Janu Remešovi za spolupráci a konzultace při vývoji mobilní aplikace.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, dne 5. května 2014

Tomáš Frič

Podpis

Abstrakt

Cílem této práce bylo seznámení se s problematikou měření znečištění ovzduší, konkrétně pak měření koncentrace polétavých částic. Za tímto účelem bylo dále cílem vytvořit návrh a prototyp hardwarového prostředku pro měření koncentrace prachových částic v ovzduší, navrhnout a naprogramovat mobilní aplikaci pro platformu Android, která by vizualizovala výsledky lokálního i vzdáleného měření (data naměřená jinými zařízeními) a realizovat komunikační rozhraní mezi hardwarovým prostředkem a aplikací pomocí Bluetooth. Nedílnou součástí práce bylo také otestování celého systému v běžném provozu a to po dobu jednoho týdne.

The goal of this thesis was to familiarize with the air pollution measurement issue, specifically with the particulate matter concentration measurement. For this purpose, the goal was to create a design and a prototype of hardware for measurement of the particulate matter concentration, program a mobile application for the Android platform that would visualize the results of both the local and remote measurements (data from other similar devices) and implement communication interface between the application and the hardware using Bluetooth. Inseparable part of this thesis was to test the complete system in an application environment for at least one week.

Obsah

1 Úvod 1
1.1 Projekt „kanárci“ 1
1.2 Účinky polétavého znečištění 2
1.2.1 Účinky na lidské zdraví 3
1.2.2 Další účinky polétavého znečištění 4
1.2.2.1 Účinky na zemské klima 4
1.2.2.2 Účinky na prostředí 5
1.3 Regulace polétavého znečištění 5
1.3.1 Evropské emisní limity 5
1.3.2 Limitní hodnoty polétavého znečištění 7
1.4 Rešerše existujících aplikací 8
1.4.1 AirCasting 8
1.4.2 Birdi 9
1.4.3 AirPi 9
1.4.4 Air Quality Egg 10
2 Návrh hardwarové části 11
2.1 Úvodní studie 11
2.1.1 Bluetooth 11
2.1.2 Napájení 14
2.1.3 Mikrokontroler 15
2.2 Použité součástky 17
2.3 Zapojení obvodu 22
2.3.1 Napájecí obvod 23
2.3.2 Měřící obvod 23
2.3.3 Komunikační obvod 24
2.4 Konfigurace obvodu 25
2.4.1 Konfigurace modulu BlueSMiRF Silver 25
2.4.1.1 Tovární nastavení 25
2.4.1.2 Změna nastavení 26
2.4.2 Programování MCU 29
2.4.2.1 Vývojová prostředí a firmware 29
2.4.2.2 Programování MCU prostřednictvím MSP-EXP430G2 32
3 Návrh mobilní aplikace 34
3.1 Funkční požadavky 34
3.2 Vývojové prostředí 35
3.2.1 Použitý software 35
3.2.2 Použité knihovny 36
3.2.2.1 Použití knihoven v Eclipse IDE 37
3.2.3 Minimální požadavky 38

3.3 Uživatelské rozhraní 39
3.3.1 Použité pojmy 39
3.3.2 Architektura UI 41
3.3.2.1 Fragment Měření 41
3.3.2.2 Fragment Mapa 43
3.3.2.3 Fragment Legenda 45
3.3.2.4 Fragment Statistiky 46
3.4 Implementace funkcí aplikace 47
3.4.1 Databáze 47
3.4.2 Komunikace se serverem 49
3.4.3 Zobrazení mapy 50
3.4.4 Komunikační rozhraní Bluetooth 52
3.4.5 Zjištění souřadnic GPS 54
3.4.6 Kreslení grafů ve Fragmentu Statistiky 55
4 Testování 57
5 Závěr 60
5.1 Zhodnocení testování 60
5.2 Další práce a vývoj 60
5.3 Celkové zhodnocení projektu 61
6 Literatura 63

Seznam tabulek

Tabulka 1	PM10 a PM2.5.	4
Tabulka 2	Limitní hodnoty polétavého znečištění pro Evropskou Unii.	6
Tabulka 3	Maximální roční a denní limity koncentrace polétavých částic.	8
Tabulka 4	Podíl verzí OS Android na mobilních zařízeních.	12
Tabulka 5	Operační charakteristiky obvodu LM2574N-5 a LM2574N-3.3	18
Tabulka 6	Operační charakteristiky prachového senzoru GP2Y1010AU0F.	19
Tabulka 7	Operační charakteristiky Bluetooth modulu BlueSMiRF Silver a RN-42.	20
Tabulka 8	Operační charakteristiky MCU M430G2553.	21
Tabulka 9	Stupně znečištění ovzduší polétavými částicemi.	42
Tabulka 10	Stupně znečištění pro látky měřené stanicemi ČHMÚ.	51

Seznam obrázků

Obrázek 1	Mapa kombinovaného venkovského a městského znečištění částicemi PM10.	7
Obrázek 2	Podíl operačních systémů v telefonech prodaných od Q1 2007 do Q4 2013.	14
Obrázek 3	Regulátor napětí LM2574N-5.	17
Obrázek 4	Prachový senzor Sharp GP2Y1010AU0F.	18
Obrázek 5	Bluetooth modul Sparkfun BlueSMiRF Silver s modulem RN-42.	19
Obrázek 6	BlueSMiRF Silver podbrobněji.	20
Obrázek 7	MCU TI M430G2553 v patici MSP-EXP430G2.	21
Obrázek 8	Schéma zapojení prototypu mobilního Kanárka.	22
Obrázek 9	Zapojení napájecího obvodu v prototypu mobilního Kanárka.	23
Obrázek 10	Zapojení prachového senzoru v prototypu mobilního Kanárka.	24
Obrázek 11	Zapojení modulu BlueSMiRF Silver v prototypu mobilního Kanárka.	24
Obrázek 12	Zapojení pro nastavení modulu RN-42 prostřednictvím vývojového kitu Freaduino UNO.	27
Obrázek 13	MSP-EXP430G2 LaunchPad.	32
Obrázek 14	Zapojení pro programování MCU M430G2xx prostřednictvím MSP-EXP430G2 LaunchPad mimo patici LaunchPadu.	33
Obrázek 15	Příklad Pevných Tabů ve dvou barevných schématech.	39
Obrázek 16	Příklad Lišty aplikace (ActionBar).	40
Obrázek 17	Fragment Měření při nezměřené hodnotě.	43
Obrázek 18	Fragment Měření při naměřené hodnotě.	43
Obrázek 19	Fragment Mapa.	44
Obrázek 20	Fragment Legenda.	45
Obrázek 21	Fragment Statistiky zobrazující denní výpis.	46
Obrázek 22	Fragment Statistiky zobrazující týdenní výpis.	46
Obrázek 23	Sestrojený prototyp mobilního Kanárka, na kterém proběhlo testování.	57
Obrázek 24	Statistika testovacího týdne.	58
Obrázek 25	Statistika testovacího dne 2.5.2014	58
Obrázek 26	Statistika testovacího dne 3.5.2014	58
Obrázek 27	Statistika testovacího dne 4.5.2014	58
Obrázek 28	Statistika testovacího dne 5.5.2014	59
Obrázek 29	Statistika testovacího dne 6.5.2014	59
Obrázek 30	Statistika testovacího dne 7.5.2014	59
Obrázek 31	Statistika testovacího dne 8.5.2014	59

Seznam příloh

Obsah přiloženého CD:

workspace/
Složka s workspace, vytvořeným v Eclipse IDE Kepler. Obsahuje Android projekt *Kanarci_Android*. Workspace lze otevřít v Eclipse IDE zvolením *File → Switch Workspace → Other...* . Je nutné nainstalovat ADT plugin a Android SDK.

google-play-services_lib/
Složka s knihovním Eclipse projektem Google Play Services.

GraphView-master/
Složka s knihovním Eclipse projektem GraphView.

Kanarek_energia/
Složka s projektem pro MCU M430G2553 *Kanarek_energia*. Tento projekt lze otevřít v IDE Energia.

Návrh aplikace pro měření znečištění ovzduší – Tomáš Frček.pdf
Tento dokument ve formátu PDF.

1 Úvod

1.1 Projekt 'Kanárci'

Projekt *Kanárci* vznikl v na Social Inovation Camp 2012. V roce 2013 vzniklo občanské sdružení s cílem zvýšení veřejného povědomí o znečištění ovzduší a jeho dopadech na zdraví. Dalším cílem bylo sestrojení zařízení pro měření znečištění ovzduší a vizualizaci těchto výsledků pomocí *Kanárka* – analogie s kanárky v klecích, které s sebou horníci brali do dolů. Tento nápad se setkal se zájmem například ze strany školek.

Znečištění ovzduší je měřeno Českým Hydrometeorologickým Ústavem. Stanice ČHMÚ ale nejsou rozmístěny v dostatečné koncentraci pro adekvátní monitoring znečištění ovzduší. *Kanárci* mají za cíl tyto mezery co nejvíce zaplnit rozšířením projektu mezi co nejvíce lidí [9].

Kanárek je spojení mikrokontroleru, prachového senzoru a komunikačního rozhraní. V současné době je použit open-source hardware Arduino nebo IOIO. Existují dva typy *Kanárků*:

- Stacionární *Kanárek* je určen pro dlouhodobé sledování kvality ovzduší v jednom místě. Je napájen ze sítě a komunikuje prostřednictvím LAN technologie Ethernet a to jak drátově tak i bezdrátově s použitím WiFi.
- Mobilní *Kanárek* je napájený bateriově. Komunikuje prostřednictvím technologie Bluetooth nebo přes softwarový modem a audio port mobilního telefonu. Mobilní *Kanárek* neodesílá naměřená data přímo na server, ale komunikuje s mobilní aplikací, která výsledky vizualizuje a data dále zpracovává a odesílá.

V současné době je velká většina aktivních *Kanárků* stacionárních. Mobilní *Kanárci* jsou stále ve vývoji. Mobilní aplikace pro platformu iOS je také hotová.

Cílem této práce je navrhnout a sestrojit prototyp mobilního *Kanárka*, který komunikuje pomocí Bluetooth, dále je cílem navrhnout a naprogramovat mobilní aplikaci pro platformu Android, která komunikuje s navrženým *Kanárkem* a zpracovává data, která z něj příjme. Hlavním omezením je cena sestrojení *Kanárka*. Horní limit ceny použitých součástek byl stanoven na 100 Euro.

Mikrokontroler (**MCU**) je malý počítač na jediném integrovaném čipu. Obsahuje procesor, paměť a programovatelné vstupní a výstupní periferie. Také obsahuje FLASH paměť pro uchování programu a jeho zavedení po resetu nebo přerušení napájení. Mikrokontrolery jsou určeny pro realizaci vestavěných systémů [10].

Typické periferie obsažené v MCU zahrnují obvod UART (hardwarový UART), A/D a D/A převodníky, časovače a paměť RAM. Další periferie mohou být například řadič LCD displaye, Ethernet, USB nebo PWM generátor.

Typickými výrobci mikrokontrolerů jsou například ST Microelectronics a Texas Instruments. Mikrokontrolery jsou typicky osazeny slabším procesorem s velice nízkým odběrem elektrického proudu a pamětí s menší kapacitou. Důvodem je to, že vestavěné systémy jsou určeny k vykonávání jednoduchého řídícího programu. Mnohdy jsou tyto systémy napájeny bateriově, nízká spotřeba je proto nezbytností. Alternativou jsou takzvané systémy na čipu (System on Chip – SoC). SoC jsou určeny pro výkonnější procesory, disponují mnohem větší pamětí a mohou být i víceprocesorové. SoC jsou schopné fungovat jako platforma pro embedded verze operačních systémů Windows nebo Linux.

Softwarový modem (**softmodem**) je modem, který je realizovaný s použitím minima vlastního hardware. Namísto toho softmodem využívá software a zdroje systému, ve kterém hostuje. Výhoda softmodem oproti hardwarovému modemu je velikost, cena a energetické nároky. Protože nemají žádnou naprogramovanou funkci, software systému, na kterém hostují, je může využít k jiným funkcím než jako klasický modem, například ke generování signálu [12].

Této vlastnosti softmodemu je využito právě u některých *Kanárků*. Komunikace mezi *Kanárkem* a aplikací je realizována tak, že firmware *Kanárka* pomocí softmodemu odešle zakódovaná data (data z prachového senzoru) prostřednictvím audio linky do aplikace, která data zpracovává a přeposílá dál (aplikace v mobilním telefonu, nebo počítači). Aplikace tato data dekóduje a dále s nimi pracuje.

1.2 Účinky polétavého znečištění

V této kapitole jsou stručně popsány škodlivé dopady polétavých částic na lidské zdraví, v krátkosti přiblíženy další oblasti, které polétavé znečištění negativně ovlivňuje a vysvětlen samotný pojem *polétavé částice*.

1.2.1 Účinky na lidské zdraví

Negativní vliv polétavých částic na lidské zdraví je přímo úměrný jejich velikosti. Největší nebezpečí představují částice o velikosti menší než 10 mikrometrů (PM10). Takto malé částice se mohou dostat hluboko do plic a některé dokonce i přímo do krevního oběhu. Vystavení těmto částicím může tedy poškodit jak plíce, tak i srdce. Znečištění polétavými částicemi obsahuje mikroskopické částice v pevném i kapalném skupenství, které se díky své velikosti mohou dostat hluboko do dýchacího ústrojí a způsobit vážné zdravotní potíže. Mezi tyto potíže se řadí například:

- Předčasné úmrtí u osob s onemocněním srdce a plic
- Lehký infarkt
- Srdeční arytmie
- Zhoršené astma
- Snížená funkčnost plic
- Respirační potíže (podrážděné dýchací cesty, kašel, ztížené dýchání)
- Kardiovaskulární potíže
- Rakovina plic
- Vrozené vady

Nejvíce ohroženou skupinou jsou lidé s onemocněním srdce a plic, děti a senioři. Při zvýšeném znečištění polétavými částicemi však může dočasné symptomy pocítit i zdravý člověk [1]. Znečistění ovzduší polétavými částicemi přispívá každoročně k předčasné smrti 22.000 – 52.000 lidí ve Spojených Státech Amerických (od roku 2000) a od roku 2005 způsobilo předčasnou smrt 370.000 lidí v Evropě [2].

Polétavé částice (particulate matter – **PM**) nebo také **polétavé znečištění** (particle pollution), případně **prachové částice** je směs extrémně malých částic a kapének. Polétavé znečištění se skládá z mnoha komponent včetně kyselin (dusičnany a sírany), organických chemikálií, kovů, půdních a prachových částic. Velikost polétavých částic je přímo úměrná zdravotním rizikům, která způsobují [1]. Polétavé částice se podle velikosti dělí do dvou skupin [3]:

PM10 – částice o velikosti od 2,5 do 10 mikrometrů (25x – 100x tenčí než lidský vlas). Tyto částice jsou méně nebezpečné, protože jsou příliš velké na to, aby pronikly tenkými dýchacími cestami hlouběji do plic.

PM2.5 – menší částice o velikosti méně než 2,5 mikrometrů. Tyto částice jsou velice nebezpečné, protože dokáží proniknout drobnými dýchacími cestami hluboko do plic. Zároveň jejich složení je více toxické, než složení částic PM10.

	Složení	Vznik	Dosah
PM10	Kouř, prach z továren, farem a cest, rostlinné výtrusy, pyl	Drcení skal a půdy, poté rozfoukané větrem	100 m – 50 km (ve vzduchu v řádech minut až hodin)
PM2.5	Toxické organické sloučeniny, těžké kovy	Automobilová doprava, pálení rostlin (lesní požáry, pálení listí...), zpracování kovů	Stovky kilometrů (ve vzduchu v řádech dnů až týdnů)

Tab. 1 PM10 a PM2.5 [3].

1.2.2 Další účinky polétavého znečištění

V této kapitole jsou stručně přiblíženy další důležité oblasti, které jsou ovlivňovány polétavým znečištěním.

1.2.2.1 Účinky na zemské klima

V souvislosti s klimatem se směs polétavého znečištění a vzduchu označuje jako *aerosol*. Aerosoly ovlivňují zemské klima tak, že mění množství solární radiace, která vstupuje do atmosféry a množství radiace, která je v atmosféře uchována. Aerosoly v atmosféře jsou zdrojem největší nejistoty v předpovědích klimatických změn – na rozdíl od skleníkových plynů, jejichž vliv může být určen s poměrně velkou přesností [2].

1.2.2.2 Účinky na prostředí

Polétavé znečištění ovlivňuje prostředí následujícími způsoby [1]:

- Zhoršená viditelnost – je způsobená hlavně menšími částicemi PM2.5, projevuje se oparem nebo mlhou.
- Poškození prostředí – polétavé částice mohou být přenášeny větrem na dlouhé vzdálenosti, poté se usazují na zem, nebo na vodní hladinu. To způsobuje například kyselost vodních ploch a toků, snížené množství živin v půdě, znehodnocování citlivých zemědělských rostlin a narušování ekosystémů.
- Estetické poškození – polétavé znečištění může poškodit kamenné stavby, například sochy a monumenty.

1.3 Regulace polétavého znečištění

V této kapitole jsou uvedeny limitní hodnoty znečištění polétavými částicemi zavedené několika světovými průmyslovými mocnostmi. Dále jsou zde krátce představeny Evropské emisní standardy, které regulují polétavé a další znečištění, způsobené automobilovou dopravou v zemích Evropské Unie. Silniční doprava je v Evropské Unii hlavním zdrojem znečištění ovzduší [5].

1.3.1 Evropské emisní standardy

Evropské emisní standardy regulují emise osobních a nákladních automobilů, vlaků a traktorů. Neregulují však emise velkých lodí a letadel. V současné době se regulují emise oxidů dusíku (No_x), uhlovodíků (total hydrocarbon – THC), uhlovodíků vyjma methanu (non-methane hydrocarbon – NMHC), oxidu uhelnatého (CO) a polétavých částic (PM). Automobily, nesplňující standardy, se nesmí v zemích Evropské Unie prodávat, standardy se však nevztahují na vozidla, která jsou již v provozu [6].

V současnosti evropské emisní standardy rozlišují 6 kategorií (Euro I – Euro VI). Každá z těchto

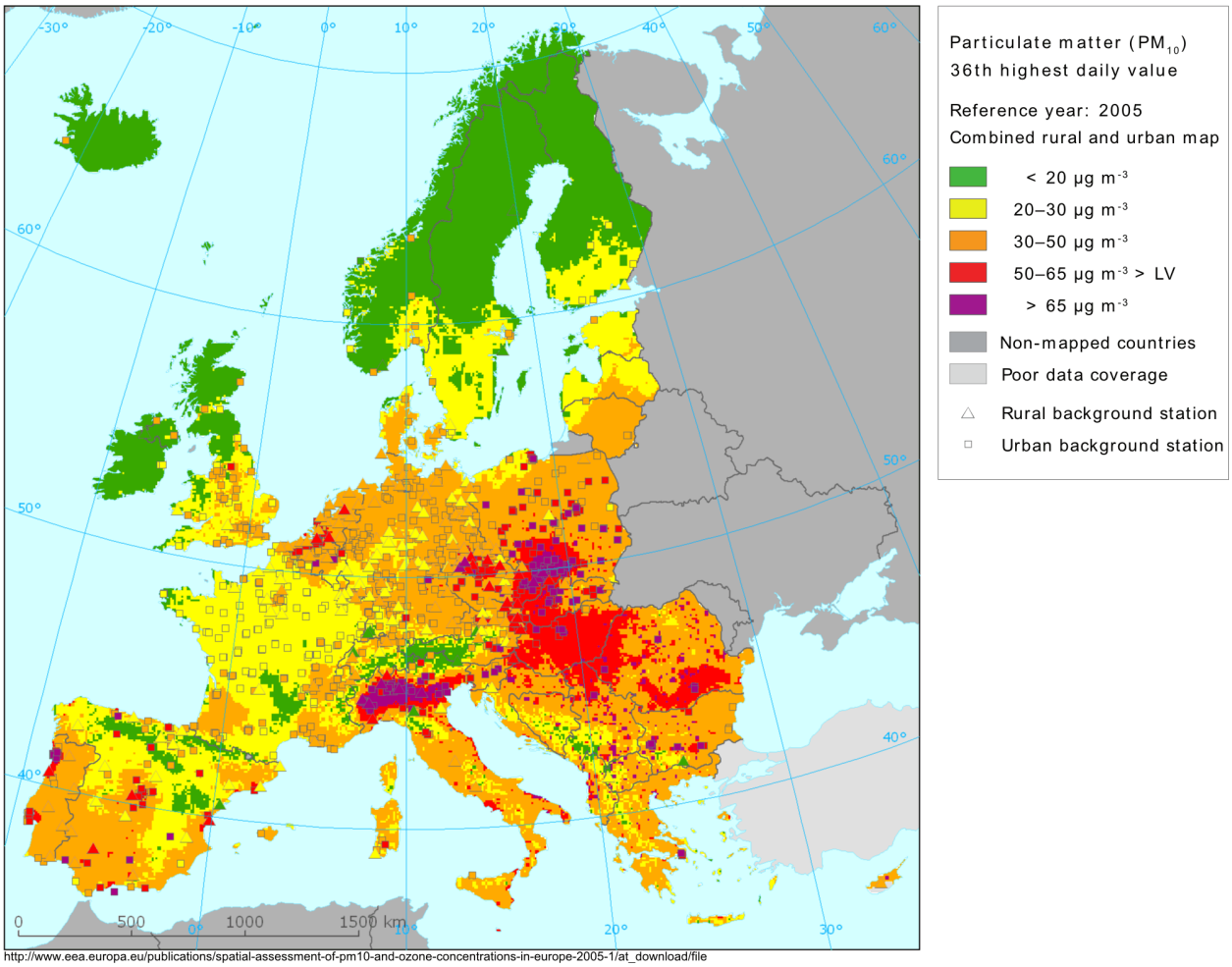
kategorií zavádí různé emisní limity pro každou z těchto kategorií motorových vozidel: osobní automobily, lehké nákladní automobily (podkategorie podle hmotnosti: ≤ 1305 kg, 1305 kg – 1760 kg, 1760 kg – 3500 kg), kamiony a autobusy a těžká nákladní vozidla [6].

Podrobnosti o limitních hodnotách a legislativě evropských emisních standardů pro silniční dopravu jsou k dispozici na stránkách Evropské komise [7].

Podrobnosti o uzákoněných standardech kvality ovzduší a limitních hodnotách znečišťujících látek a polétavého znečištění jsou k dispozici na stránkách Evropské komise [7].

	PM10 Od 1.1.2005	PM2.5 Od 1.1.2015
Roční průměr	40 μg/m^3	25 μg/m^3
Denní průměr	50 μg/m^3	-
Povolené množství překročení limitu za rok	35	-

Tab. 2 Limitní hodnoty polétavého znečištění pro Evropskou Unii [2].



Obr. 1 Kombinovaná mapa venkovského a městského polétavého znečištění částicemi PM10. Data z roku 2005 [8].

1.3.2 Limitní hodnoty polétavého znečištění

Následující tabulka znázorňuje stanovené limity polétavého znečištění v několika průmyslových a znečištění produkujících zemích. Čísla v závorkách značí maximální povolený počet překročení limitů za jeden rok. Pokud není uvedeno, tolerance je nulová. [2]:

Země	PM10		PM2.5	
	Max. roční průměr	Max. denní průměr	Max. roční průměr	Max. denní průměr
Austrálie	-	50 μg/m^3	8 μg/m^3	25 μg/m^3
Čína	70 μg/m^3	150 μg/m^3	35 μg/m^3	75 μg/m^3
EU	40 μg/m^3	50 μg/m^3 (35)	25 μg/m^3	-
Hong Kong	50 μg/m^3	100 μg/m^3 (9)	35 μg/m^3	75 μg/m^3 (9)
Japonsko	-	100 μg/m^3	15 μg/m^3	35 μg/m^3
Jižní Korea	50 μg/m^3	100 μg/m^3	25 μg/m^3	50 μg/m^3
Spojené Státy	-	150 μg/m^3 (1)	15 μg/m^3	35 μg/m^3

Tab. 3 Maximální roční a denní limity koncentrace polétavých částic.

1.4 Rešerše existujících aplikací

Existuje více projektů zaměřených na kontrolu kvality ovzduší. Každý z nich je něčím specifický. Projekt *Kanárci* se zaměřuje na sledování koncentrace polétavých částic v ovzduší, které znamenají veliké zdravotní riziko, zatímco některé další projekty sledují jiné znečišťující prvky. Další specifikací *Kanárků* je cena. Zatímco existující prototypy jsou založeny na vývojových kitech Arduino nebo IOIO, cílem této práce je navržení a implementace levnějšího řešení. Jako cenový strop byla stanovena cena 100 Euro za použité součástky. Následuje krátký přehled aplikací pro sledování kvality ovzduší a jejich srovnání s *Kanárky*.

1.4.1 AirCasting

AirCasting se skládá z aplikace pro OS Android a monitorovacího zařízení pro měření znečištění ovzduší (základem zařízení je Arduino kit). Aplikace se zařízením komunikuje pomocí Bluetooth. AirCasting umožňuje sledovat hladinu hluku pomocí mikrofonu telefonu a teplotu, vlhkost, CO a NO_2 pomocí měřícího zařízení. AirCasting dále umožňuje sledování fyziologických veličiny, jako srdeční činnost, dýchání nebo tělesnou teplotu pomocí modulů Zephyr BioHarness 3 a Zephyr HxM. Sesbíraná data jsou sdílena s dalšími uživateli prostřednictvím mapy s vyznačenými oblastmi a

koncentracemi naměřených látek [28].

Oproti *Kanárkům* je řešení AirCasting podstatně dražší. Aplikace je zdarma dostupná na Google Play, postavení monitorovacího zařízení na kitu Arduino je ale mnohem nákladnější než řešení *Kanárků*. Pro sledování fyziologických veličin je také nutné zakoupit potřebná zařízení od třetí strany. AirCasting také nenabízí sledování polétavého znečištění.

1.4.2 Birdi

Birdi je stacionární zařízení, určené pro umístění v bytech a rodinných domech. Jeho hlavní funkcí je monitorování koncentrací kouře a CO_2, funguje tedy primárně jako detektor ohně. Při detekci zvýšených koncentrací pošle zařízení Birdi notifikaci do mobilní aplikace, kterou je nejprve nutné se zařízením synchronizovat pomocí WiFi. V případě nebezpečí kontaktuje Birdi hasiče a majitele dalších zařízení Birdi v blízkém okolí. Další funkcí zařízení Birdi je monitorování teploty, vlhkosti a znečištění ovzduší [29].

Birdi je určeno pouze pro stacionární použití, neexistuje žádná mobilní verze. Vzhledem k určení Birdi jako detektoru kouře v domech a bytech je neexistence mobilní verze pochopitelná a nijak nelimituje celkovou funkcionalitu projektu. Oproti Birdi jsou *Kanárci* určeny jak ke stacionárnímu, tak i mobilnímu monitorování hladiny polétavého znečištění.

1.4.3 AirPi

AirPi je shield kit pro Rospberry Pi. Je určen pro monitorování atmosferického tlaku, teploty, vlhkosti, světla, kouře a částic CO a NO_2. Zařízení je možné si podle návodu sestavit doma. Na oficiálních stránkách projektu uvádějí cenu 55 liber za pořízení všech senzorů (je možné koupit a připojit také pouze vybrané senzory, není třeba připojit všechny). Tato cena však nezahrnuje koupit samotného Rospberry Pi [30].

Hotové zařízení je poměrně objemné a je určené ke stacionárnímu použití. Výsledky měření jsou odesílány na internetový server, neexistuje mobilní aplikace, se kterou by zařízení bylo možné

synchronizovat.

1.4.4 Air Quality Egg

Air Quality Egg je systém navržený ke sledování koncentrací CO a NO_2. Celý systém se skládá z venkovních senzorů, které bezdrátově posílají naměřená data do „Egg“ stanice, typicky uvnitř domu. Egg stanice funguje jako uživatelské rozhraní, prostřednictvím LED diod indikuje stav systému. Dále přeposílá naměřená data na internetový server Xively [31].

Toto řešení je opět stacionární a monitoruje pouze koncentrace CO a NO_2. *Kanárci* jsou zaměřené primárně na monitorování polétavého znečištění.

2 Návrh hardwarové části

Tato práce je rozdělena do dvou částí. V první části je popsán návrh samotného *Kanárka*. Je diskutován způsob návrhu, výběr součástek a limitující faktory. Jsou popsány použité součástky, důvod jejich výběru a jejich zapojení do finálního obvodu. Dále je vysvětlen způsob programování MCU, které řídí celého *Kanárka*. Ve stručnosti je představeno použité MCU, programovací prostředí a programovací jazyk.

2.1 Úvodní studie

V této kapitole jsou diskutovány možné způsoby řešení návrhu mobilního *Kanárka*, problematické části výběru použitých technologií a součástek a z nich vzešlé kompromisy.

2.1.1 Bluetooth

Technologie Bluetooth je použita pro odesílání naměřených hodnot prachovým senzorem do mobilní aplikace. Při návrhu komunikačního protokolu byla hlavním limitujícím faktorem spotřeba energie použitého Bluetooth modulu. Snaha o minimalizaci spotřeby pramenila z faktu, že mobilní *Kanárek* je napájen bateriově. Z tohoto důvodu se tedy nabízelo použití Bluetooth verze 4. Bluetooth 4 zavedlo novou kategorii této technologie označovanou jako Bluetooth Low Energy (BLE). Jiné označení je Bluetooth Smart. BLE je určena pro aplikace s velmi nízkým proudovým odběrem. Oproti Bluetooth 2 má kratší dosah (stále však dostačujících 50 m) a nižší přenosovou rychlost (0,27 Mbps, pro potřeby *Kanárka* naprosto postačující). Na rozdíl od Bluetooth 2 technologie, která s bateriovým napájením vydrží řádově dny, BLE vydrží řádově měsíce až roky. Bluetooth 4 přináší i další pokroky, například odpadá potřeba párování zařízení.

Komplikace ale nastala na straně mobilní aplikace. Přestože většina nových telefonů s operačním systémem Android má hardwarovou podporu BLE, Android podporuje BLE až od API verze 18

(Android 4.3 Jelly Bean). Protože při vývoji a testování byl k dispozici pouze telefon s OS Android 4.2.2 (API verze 17), byla pro prototyp zvolena varianta s Bluetooth 2. Protože Bluetooth modul v obvodu pracuje v režimu Serial Port Profile (SPP), ve kterém bezdrátově nahrazuje rozhraní RS-232, budoucí upgrade na Bluetooth 4 bude zahrnovat pouze výměnu Bluetooth modulu a modifikaci kódu mobilní aplikace.

Verze	Označení	API	Podíl
2.2	Froyo	8	1,1 %
2.3.3 – 2.3.7	Gingerbread	10	17,8 %
3.2	Honeycomb	13	0,1 %
4.0.3 – 4.0.4	Ice Cream Sandwich	15	14,3 %
4.1.x	Jelly Bean	16	34,4 %
4.2.x		17	18,1
4.3		18	8,9 %
4.4	KitKat	19	5,3 %

Tab. 4 Podíl verzí OS Android na mobilních zařízeních, data za 7 dní s koncem 1. dubna 2014. Z tabulky lze vidět, že podíl zařízení s podporou Bluetooth 4 (API 18+) k tomuto datu byl pouhých 14,2 % [13].

Bluetooth je standard bezdrátové komunikace určené pro krátké vzdálenosti. Využívá radiové vlny s krátkou vlnovou délkou a ultra vysokými frekvencemi (Ultra High Frequency – UHF) o kmitočtech pohybujících se v rozmezí 2,4 Ghz až 2,485 Ghz. Technologie byla vyvinuta firmou Ericsson v roce 1994 a původně byla určena jako bezdrátová verze rozhraní RS-232. V současnosti je Bluetooth spravován skupinou Bluetooth Special Interest Group (SIG). Aby zařízení obdrželo označení 'Bluetooth zařízení', musí splnit standardy vytyčené SIG [17].

Bluetooth je určen pro realizaci bezdrátové komunikace s nízkou spotřebou energie. Dosah komunikace je závislý na třídě výkonu (power class), ve které zařízení pracuje. Bluetooth rozlišuje 3 třídy výkonu [17]:

Třída	**maximální výkon [mW]**	**maximální výkon [dBm]**	**dosah [m]**
1	100	20	100
2	2,5	4	10
3	1	0	1

Bluetooth byl původně navržen jako bezdrátová náhrada kabelu pro RS-232, tedy pro sériovou komunikaci. Dnes existují různé profily komunikace, podobně jako třídy zařízení u standardu USB. Pro potřeby *Kanárka* je využit Serial Port Profile (SPP). SPP emuluje sériovou komunikaci založenou na standardu RS-232.

Existují 4 verze Bluetooth [17]:

Bluetooth 1

První verze technologie Bluetooth. Obsahovala mnoho chyb a propojení zařízení bylo mnohdy problematické. Verze 1.1 a 1.2 mnoho chyb opravily, zvýšily rychlost spojení i přenosu. Verze 1 nabízí rychlost dat až 1 Mbit/s.

Bluetooth 2 + EDR

Tato verze byla představena v roce 2004 a zahrnovala Enhanced Data Rate (EDR), která zvyšuje rychlost přenosu až na 3 Mbit/s (v praxi většinou kolem 2,1 Mbit/s).

Bluetooth 3 + HS

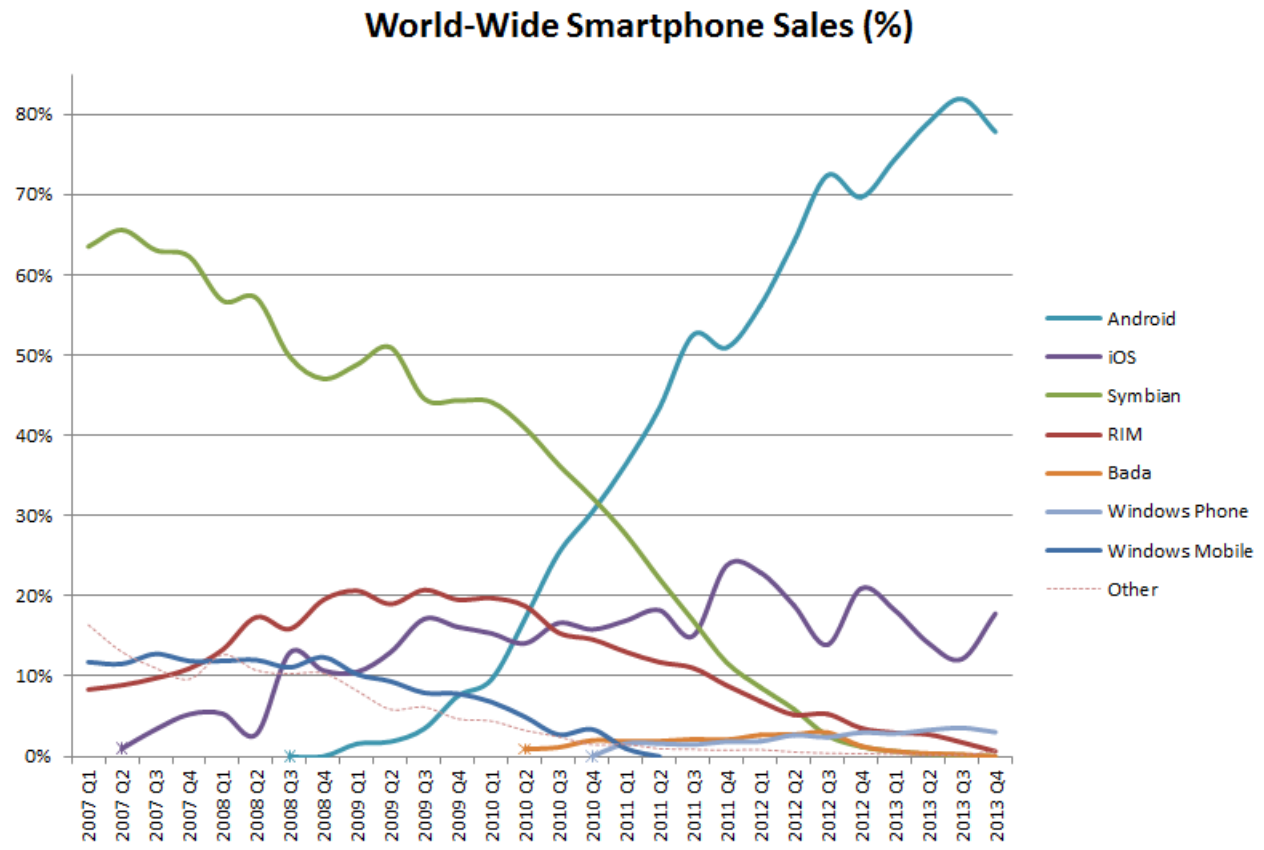
Představena v roce 2009. High Speed (HS) Bluetooth nabízí rychlost až 24 Mbit/s.

Bluetooth 4 (Bluetooth Smart, Bluetooth Low Energy)

Představena v roce 2010. Teoretická rychlost dat 24 Mbit/s. Hlavní změnou je zavedení standardu Bluetooth Low Energy (BLE), který dramaticky snižuje energetické nároky bezdrátové komunikace.

Android je operační systém založený na jádře Linux, určený pro zařízení s dotykovým displayem (chytré telefony a tablety). Android je od roku 2005 vlastnictvím společnosti Google. První smartphone s OS Android, HTC Dream, byl dostupný 22. října 2008. Od konce roku 2010, kdy na prvním místě vystřídal Symbian, je Android jasně dominantním operačním systémem v mobilním světě [14].

Android je open source projekt. Vývojář může neomezeně využívat veškeré základní funkce telefonu (například posílání zpráv, volání nebo fotoaparát). Systém je navržen tak, aby nerozlišoval mezi základními aplikacemi a aplikacemi třetích stran ve smyslu oprávnění k přístupu ke zdrojům telefonu [15].



Obr. 2 Podíl operačních systémů v telefonech prodaných od prvního čtvrtletí 2007 do čtvrtého čtvrtletí 2013 [14].

2.1.2 Napájení

V návrhu se nacházejí obvody s různými hodnotami napájecího napětí. Prachový senzor a Bluetooth modul jsou napájeny napětím 5 V. Vybrané MCU je navrženo jako nízkoenergetické a je napájeno napětím 3,3 V. Pro prototyp bylo tedy potřeba vytvořit dvě napájecí větve (5 V a 3,3 V). Jako řešení tohoto problému bylo zvoleno použití dvou regulátorů napětí. Regulátor napětí je obvod, který stabilizuje vstupní napětí (které musí být větší, než požadované výstupní napětí, o kolik záleží na účinnosti regulátoru) na výstupu na požadovanou hodnotu. Jako zdroj vstupního napětí pro tyto regulátory byla pro prototyp zvolena klasická 9 V baterie.

Toto řešení je velice triviální a vyžaduje periodické výměny baterie. Další nevýhodou je dramatický pokles napětí 9 V alkalické baterie u konce její životnosti. To má za následek i pokles výstupního napětí regulátorů, což způsobuje, že součástky nejsou v takovém případě dostatečně napájeny. Pro

finální produkt je takové řešení nevhodné, pro účely prototypu ale bylo postačující.

Jako alternativa pro finální produkt se nabízí použití Lithium-ionového (Li-ion) nebo Lithium-polymerového (Li-pol) bateriového modulu. Napájení na stejné bázi mají například mobilní telefony a jiná mobilní zařízení. První výhodou je to, že lze použít 5 V verzi Li-ion (Li-pol) baterie. To odstraní potřebu použití 5 V napěťového regulátoru a v obvodu tak zůstane jediný (regulátor pro MCU na 3,3 V). 5 V Li-ion bateriový modul ale není běžně vyráběný a výroba na zakázku by v kombinaci s dalšími součástkami, nezbytnými k sestrojení obvodu, velice rychle porušila stanovený cenový strop 100 Euro. Místo toho by se dal použít 7,4 V Li-ion modul (běžně k dostání, sériová kombinace dvou klasických 3,7 V Li-ion akumulátorů). Toto řešení by vyžadovalo zachování obou napěťových regulátorů. Další výhodou je kapacita baterie. Zatímco klasické 9 V alkalické baterie disponují typickou kapacitou v řádech stovek mAh [16] Li-ion bateriové moduly mohou mít kapacitu v řádech tisíců mAh. Třetí obrovská výhoda tohoto řešení spočívá v tom, že odpadá nutnost výměny baterie. Li-ion bateriové moduly jsou dobíjecí. Do obvodu by bylo nutné zakomponovat nabíjecí obvod (se vstupem USB, nebo klasickým 5 V nebo 12 V adaptérem ze sítě). Jedinou nevýhodou je pořizovací cena. Zatímco alkalická 9 V baterie je velice levná, cena Li-ion nebo Li-pol baterie může být až v řádech tisíců korun plus cena za nabíjecí obvod. Je to ale jediná investice oproti periodickému měnění baterie, které by nutně s časem cenově předčilo navrhované řešení.

2.1.3 Mikrokontroler

Výběr MCU byl ovlivněn třemi faktory. Prvním faktorem byly periferie. Vybrané MCU muselo obsahovat následující periferie ke správné, rychlé a přesné funkci *Kanárka*:

- Analogově číslicový převodník (ADC) s dostatečně velkým rozlišením pro přesné převody analogového výstupu prachového senzoru.
- Hardwarový obvod UART (universal asynchronous receiver / transmitter) pro sériovou komunikaci. Použití softwarového UARTu vyžaduje více kódu a je pomalejší.

Dalším faktorem byla spotřeba energie. Vzhledem k povaze obvodu (bateriově napájený, s

prachovým senzorem a Bluetooth modulem s napájecím napětím 5 V a znatelným odběrem) bylo nutné vybrat MCU s co nejnižší spotřebou. MCU s výkonným procesorem (32 bit architektura), mnoha periferiemi a napájecím napětím 5 V by sice odstranilo potřebu 3,3 V napěťového regulátoru, odebíralo by ale zbytečně moc elektrického proudu a pro aplikaci *Kanárka* by bylo zbytečně výkonné. Oproti tomu MCU z řady Low Power, nabízené například společnostmi ST Microelectronics a Texas Instruments, je napájené napětím 3,3 V a je navržené speciálně s cílem minimalizovat proudový odběr. Použitím méně výkonného procesoru (8 bit nebo 16 bit architektura, rychlost procesoru do 32 MHz) se dále sníží energetické nároky obvodu přičemž se zachová dostatečný výpočetní výkon pro potřeby navržené aplikace.

Posledním faktorem bylo samotné programování MCU. MCU jsou v dnešní době velice levné (typicky se ceny pohybují maximálně ve stovkách korun u dobře vybavených MCU s velkou pamětí a 32 bit procesory, do sta korun u méně výkonných MCU). Pro účely vývoje ale bylo nutné koupit i programátor pro danou rodinu mikrokontrolerů, který už je podstatně dražší. Programátory jsou obvody s paticí a programovacím rozhraním pro danou rodinu procesorů. Pro vývoj firmware jsou dostupné také vývojové kity, které obsahují daný procesor a typicky také vyvedené piny na externí rozhraní některých periferií (GPIO nebo UART Rx a Tx piny). Tyto kity jsou určeny pouze pro vývoj a není jimi většinou možné programovat jiné MCU než to, které v sobě mají zapájené.

Pro tento projekt bylo zvoleno MCU M430G2553 od společnosti Texas Instruments (TI). Je to MCU z řady Ultra Low Power, napájené napětím 3,3 V. Obsahuje hardwarový UART obvod a 10 bit ADC. Pro programování MCU byl zvolen programátor TI MSP-EXP430G2 LaunchPad. Výhoda tohoto řešení spočívá v tom, že LaunchPad má patici pro MCU v provedení dual in-line package (DIP), která se dají dobře použít v nepájivém poli, na kterém byl prototyp sestrojen. Pomocí LaunchPadu lze také programovat MCU, která jsou již zakomponována v obvodu na nepájivém poli, není tedy nutné přemisťovat MCU mezi paticí LaunchPadu a cílovým obvodem kvůli přeprogramování.

2.2 Použité součástky

V této kapitole jsou popsány použité součástky a stručně vysvětlena jejich funkce. U každé součástky je uvedena pořizovací cena uvedená na internetovém obchodě společnosti Farnell i základní operační elektrické a další charakteristiky.

LM2574N-5, **LM2574N-3.3**

Spínaný regulátor napětí. Slouží k udržení konstantní velikosti napětí. Tyto obvody generují stejnosměrné napětí 5 V (LM2574N-5) a 3,3 V (LM2574N-3.3) ze stejnosměrného napájecího napětí o velikosti až 45 V. Garantovaný výstupní proud je 0,5 A. V obvodu jsou použity pro vytvoření 5 V a 3,3 V napájecí větve. Jejich účinnost je 77 % (5 V) a 72 % (3,3 V), napájecí napětí musí být tedy minimálně 6,5 V pro regulátor na 5 V a 4,6 V pro regulátor na 3,3 V, aby výstupní napětí bylo stabilně 5 V (V praxi se ukázalo, že ani tyto minimální hodnoty napětí nejsou dostačující a je dobré mít rezervu). Podle katalogového listu by vstupní napětí mělo být minimálně 7 V pro optimální funkci obvodu.

Obr. 3 Regulátor napětí LM2574N-5 . Pouzdro obvodu LM2574N-3.3 je stejné, liší se pouze popisek [11].

	LM2574N-5	LM2574N-3.3
U_{in}	7 V – 45 V	
U_{out}	typicky 5 V (4,9 V – 5,1 V)	typicky 3,3 V (3,234 V – 3,366 V)
I_{out}	0,5 A	
operační teplota	-40 °C – 125 °C	
frekvence	52 kHz	
účinnost	77 %	72 %
pořizovací cena	93 Kč	82 Kč

Tab. 5 Operační charakteristiky obvodů LM2574N-5 a LM2574N-3.3 podle katalogového listu.

GP2Y1010AU0F

Optický prachový senzor od firmy Sharp. Funguje na bázi Infračervené LED diody a fototranzistoru. Zaznamenává světlo odražené od prachových částic. Tento senzor je velice efektivní při detekování jemných prachových částic, například z cigaretového kouře.

Obr. 4 Prachový senzor Sharp GP2Y1010AU0F [18].

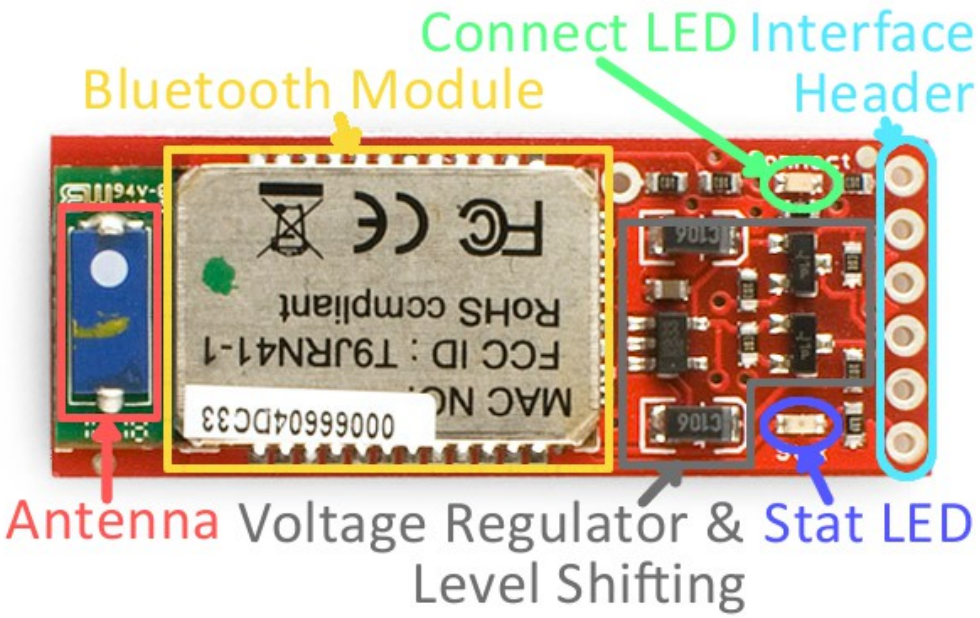
U_{cc}	5 V ± 0,5 V
I_{cc}	11 mA (maximálně 20 mA)
U_{out} (žádný prach)	0 V – 1,5 V (typicky 0,9 V)
U_{out}	3,4 V - U_{cc}
operační teplota	-10 °C – 65 °C
pořizovací cena	369 Kč

Tab. 6 Operační charakteristiky prachového senzoru GP2Y1010AU0F podle katalogového listu.

Bluetooth modul – BlueSMiRF Silver

Tento modul od firmy Sparkfun je navržen jako bezdrátová náhrada sériového kabelu. Je osazen Bluetooth modulem RN-42 (Bluetooth 2). BlueSMiRF modul funguje jako sériová roura (komunikace pomocí Rx (receive) a Tx (transmit) pinů). Napájecí napětí se může pohybovat od 3,3 V do 6 V (všechny vstupně výstupní piny modulu jsou 3 V – 6 V tolerantní, obvod obsahuje regulátor napětí, protože modul RN-42 pracuje se vstupním napětím na úrovni 3,3 V) [19]. V obvodu *Kanárka* je napájecí pin modulu připojen na 5 V větev. Modul BlueSMiRF Silver obsahuje dvě indikační LED diody, které identifikují stav modulu.

Obr. 5 Bluetooth modul Sparkfun BlueSMiRF Silver s modulem RN-42 (nahoře) [19].



Obr. 6 BlueSMiRF Silver podrobněji [19].

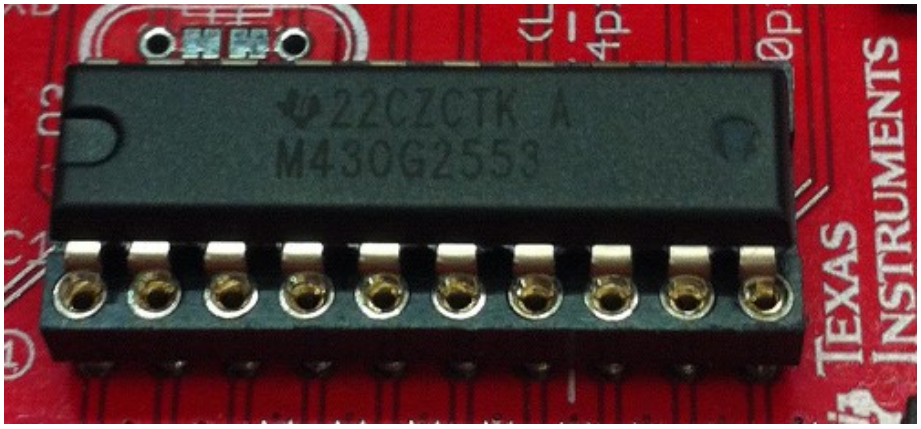
	BlueSMiRF Silver	RN-42
U_{cc}	3,3 V – 6 V	3 V – 3,6 V (typicky 3,3 V)
I_{cc} (připojeno, transfer dat)		40 mA – 50 mA (typicky 45 mA)
I_{cc} (deep sleep idle mode)		26 µA
operační teplota	-40 °C – 70 °C	-40 °C – 85 °C
pořizovací cena	$40 (e-shop společnosti Sparkfun, 800 Kč (20 Kč / USD))	

Tab. 7 Operační charakteristiky Bluetooth modulu BlueSMiRF Silver a RN-42 podle katalogového listu.

M430G2553

Mikrokontroler od firmy Texas Instruments. V obvodu má řídící úlohu, iniciuje měření, čte data ze senzoru, převádí analogová data do digitální podoby a odesílá data pomocí UART obvodu do Bluetooth modulu. MCU obsahuje 16 bit procesor s rychlostí 16 MHz a 16 bit registry. Architektura je optimalizovaná pro nízkoenergetické aplikace s cílem prodloužit životnost baterie. MCU obsahuje 16 bitové časovače, 20 programovatelných vstupně výstupních pinů, univerzální analogové

komparátory a vestavěný UART obvod pro sériovou komunikaci.



Obr. 7 MCU TI M430G2553 v patici MSP-EXP430G2 [20].

U_{cc}	1,8 V – 3,6 V
I_{cc} (active mode, 1 MHz, 2,2 V)	230 μA
I_{cc} (active mode, 1 MHz, 3 V)	330 μA (maximálně 420 μA)
I_{cc} (standby mode)	0,5 μA
operační teplota	-40 °C – 85 °C
pořizovací cena	52 Kč

Tab. 8 Operační charakteristiky M430G2553 podle katalogového listu.

Pasivní součástky

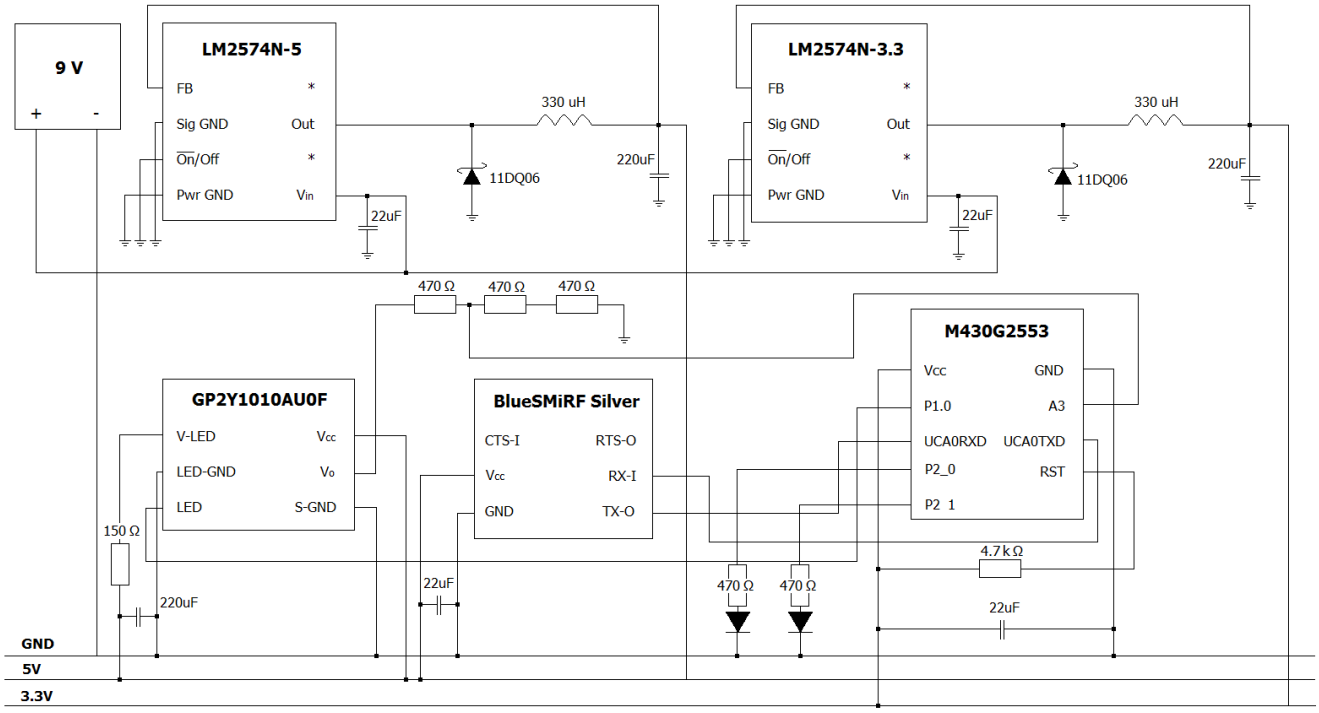
Podle schémat zapojení jednotlivých součástek (v souladu s jejich katalogovými listy) a podle obecných zásad návrhu vestavěných systému jsou k sestrojení finálního obvodu potřeba následující pasivní elektrické součástky:

- 4x elektrolytický kondenzátor 22 μF
- 3x elektrolytický kondenzátor 220 μF
- 2x induktor 330 μH

- 2x 11DQ06 Schottkyho dioda
- 1x rezistor 4,7 kΩ
- 3x rezistor 470 Ω
- 1x rezistor 150 Ω

2.3 Zapojení obvodu

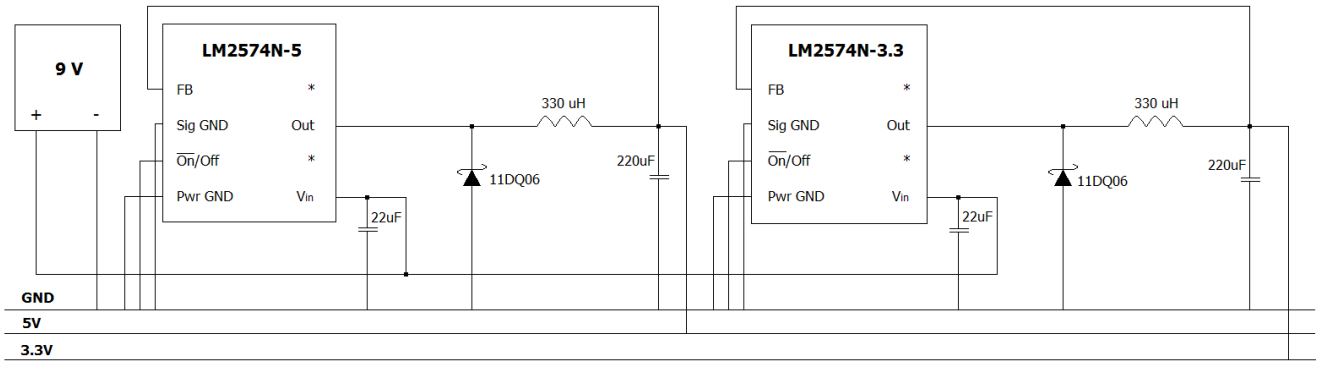
Následující schéma znázorňuje zapojení prototypu mobilního *Kanárka*. K napájení obvodu je použita 9 V alkalická baterie, ve schématu jsou popsány pouze použité piny MCU (fyzicky má MCU 20 pinů). Reset pin MCU je připojen k napájecímu pinu V_{cc} a tím stále udržován ve stavu logické 0.



Obr. 8 Schéma zapojení prototypu mobilního Kanárka.

2.3.1 Napájecí obvod

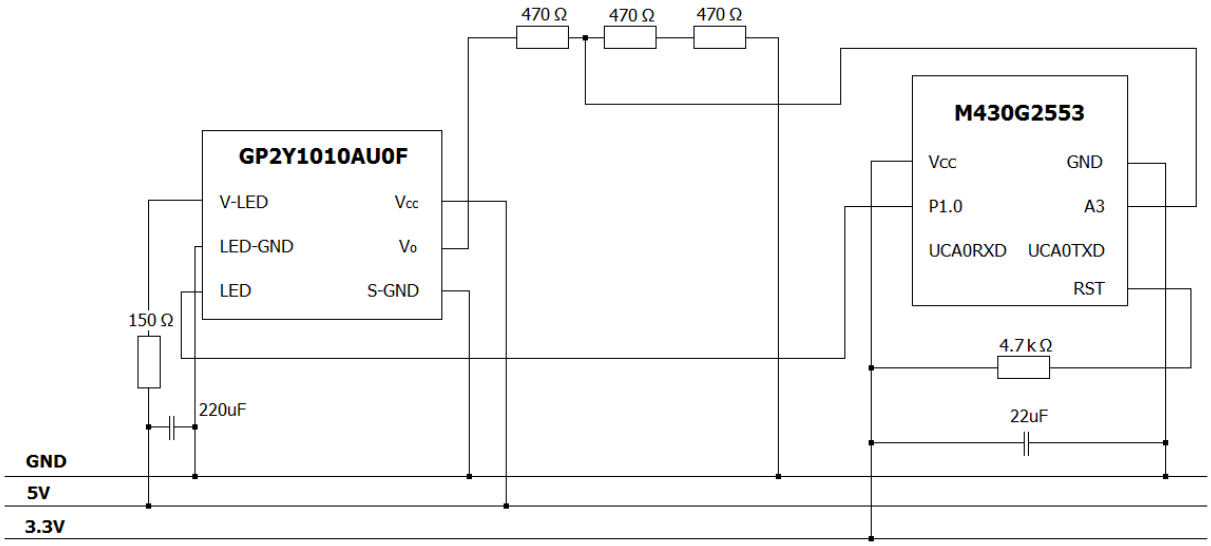
Následující schéma zobrazuje zapojení napájecího obvodu v prototypu mobilního *Kanárka*. Je využito hlavního napájecího zdroje (9 V baterie) a dvou napěťových regulátorů. Obvod je zapojen podle katalogových listů obvodů LM2574N-5 a LM2574N-3.3. Z 9 V baterie je vyvedena společná zem pro celý obvod. Výstupy regulátorů pak tvoří 5 V a 3,3 V napájecí větve.



Obr. 9 Zapojení napájecího obvodu v prototypu mobilního Kanárka.

2.3.2 Měřící obvod

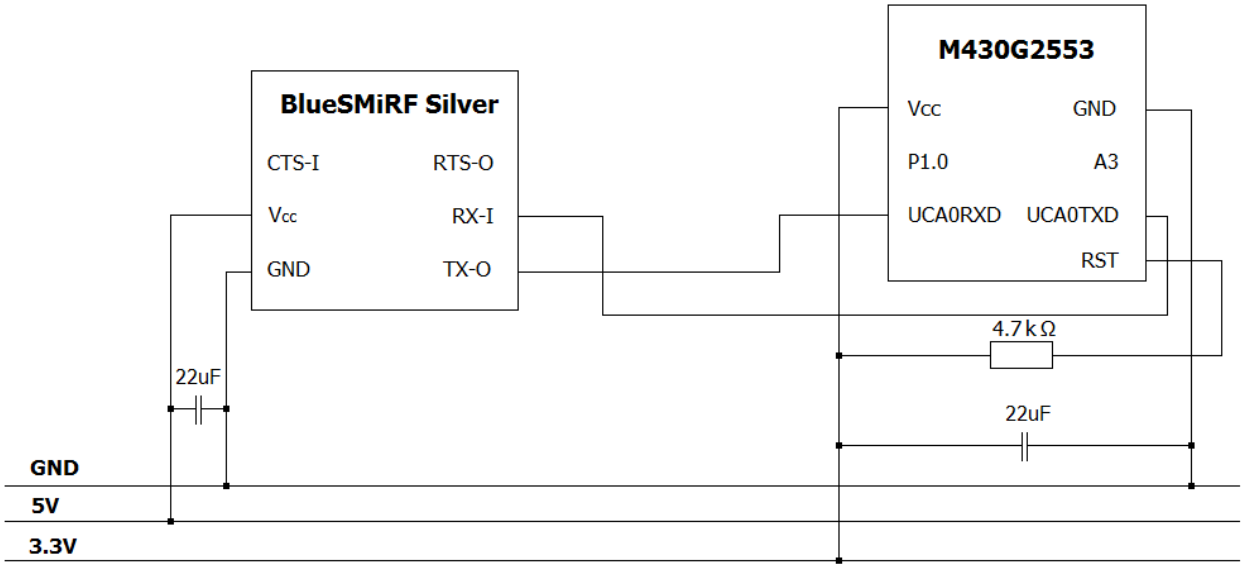
Následující schéma zobrazuje zapojení měřícího obvodu v prototypu mobilního *Kanárka*. Zobrazuje zapojení senzoru GP2Y1010AU0F a jeho připojení na piny MCU. Výstupní napětí senzoru se generuje na výstupním pinu V_o a podle katalogového listu se jeho velikost pohybuje v rozmezí 3,4 V – U_{cc} (5 V). Podle katalogového listu MSP430G2553 je ale maximální napětí, které je možné aplikovat na kterýkoliv jeho pin rovno U_{cc} (3,3 V) + 0,3 V – piny MCU jsou 5 V netolerantní. Není tedy možné připojit výstup prachového senzoru přímo na vstup ADC (pin A3). Nejprve je nutné převést úroveň výstupního napětí prachového senzoru na úroveň 3,3 V. Protože 3,3 V jsou 2/3 5 V, je snadné dosáhnout požadované úrovně aplikováním napěťového děliče pomocí rezistorů [21].



Obr. 10 Zapojení prachového senzoru v prototypu mobilního Kanárka.

2.3.3 Komunikační obvod

Následující schéma zobrazuje zapojení komunikačního obvodu. BlueSMiRF modul je nakonfigurován do profilu SPP (Serial Port Profile), ve kterém slouží jako sériová roura a pouze nahrazuje kabel pro rozhraní RS-232. RX-I a TX-O piny modulu jsou spojeny s piny UCA0TXD a UCA0RXD na MCU. Tyto piny jsou fyzicky spojeny s obvodem UART na MCU.



Obr. 11 Zapojení modulu BlueSMiRF Silver v prototypu mobilního Kanárka.

2.4 Konfigurace obvodu

V této kapitole je popsána konfigurace součástek v obvodu. Konkrétně způsob a detail konfigurace Bluetooth modulu BlueSMiRF Silver a způsob programování MCU. Dále je uveden zdrojový kód firmware MCU a vysvětlena jeho rutina.

2.4.1 Konfigurace modulu BlueSMiRF Silver

Ve skutečnosti se nekonfiguruje modul BlueSMiRF, ale modul RN-42, který je na modulu BlueSMiRF obsažen. Ke konfiguraci modul RN-42 se používá sada AT příkazů (AT (attention) commands, vychází z Hayesových příkazů vyvinutých původně ke konfiguraci modemů [22]).

2.4.1.1 Tovární nastavení

Modul RN-42 je továrně nastaven následujícím způsobem (uvedena pouze nastavení základních parametrů a parametrů relevantních pro tento projekt):

- **UUID** (**Universally unique identifier**) – nastavení důležité hlavně pro Android. Připojení k zařízení Bluetooth na straně Androidu probíhá tak, že se vytvoří instance třídy *BluetoothSocket*. K tomu je potřeba instance třídy *BluetoothDevice* (získaná vyhledáním viditelných Bluetooth zařízení nebo přímo z MAC adresy zařízení, kterou si aplikace mohla zapamatovat při předešlé komunikaci se zařízením) a UUID, se kterým se má *BluetoothSocket* vytvořit. UUID je 128 bitů dlouhá informace o protokolu nebo službě, kterou Bluetooth zařízení podporuje.

  Továrně je UUID nastaveno na profil SPP (Serial Port Profile). Toto nastavení je vyhovující pro komunikaci mezi *Kanárkem* a mobilní aplikací a mobilní aplikace ho musí použít pro úspěšné navázání spojení s modulem RN-42.

  SPP UUID: 35111C00001101000010008000008 05F9B34FB

- **Device name** – jméno Bluetooth zařízení. Toto jméno se zobrazí uživateli při hledání

Bluetooth zařízení. Továrně je toto jméno nastaveno na označení Bluetooth modulu (RN-42). Pro potřeby tohoto projektu bylo jméno zařízení nastaveno na „Kanarek9999“.

- **Pin code** – bezpečnostní kód, který modul RN-42 bude vyžadovat od každého zařízení, které se bude chtít spárovat. Pro potřeby tohoto projektu byl kód nastaven na 9999. Tovární nastavení pin kódu je 1234.
- **UART baud rate** – toto nastavení řídí rychlost přenosu dat. Tovární nastavení je 115 200 Bd/s. Pro potřeby *Kanárka* je rychlost nastavena na postačujících 9600 Bd/s.
- **Device profile** – toto nastavení řídí profil Bluetooth zařízení. Bluetooth definuje několik profilů zařízení (podobně jako USB třídy zařízení). Továrně je nastaven profil SPP, který je vyhovující pro *Kanárky*. Další profily jsou například HID (Human interface device, používané například u bezdrátových gamepadů) nebo APL (Apple profile).

Všechny změny v nastavení se projeví až po restartu modulu. Toho lze dosáhnout buď odpojením modulu od napájení, nebo softwarovým resetem v módu nastavení (příkaz „R,1“).

2.4.1.2 Změna nastavení

Některá nastavení modulu RN-42 musela být změněna (pro prototyp se jednalo o název zařízení, bezpečnostní pin kód a rychlost přenosu). Nastavení modulu RN-42 na modulu BlueSMiRF je realizováno pomocí pinů RX-I a TX-O modulu BlueSMiRF. Pro účely prototypu bylo k nastavení modulu využito vývojového kitu Freaduino UNO V1.8.1. Freaduino je fork populárního open hardware Arduino. Kit Freaduino UNO je vývojový kit poháněný 8 bitovým procesorem ATmega328. Deska také obsahuje regulátor napětí a přepínač, který řídí úroveň napětí. To umožňuje připojit 3,3 V nebo 5 V periferie přímo ke vstupně výstupním pinům kitu [23].

K programování procesoru ATmega328 na kitu Freaduino UNO je použito vývojové prostředí Arduino IDE. Programování probíhá v jazyce C++ a využívá frameworku *Wiring*. Arduino má své vlastní C++ knihovny, většinu standardních C/C++ knihoven ale nepodporuje. Důvodem je omezená velikost paměti RAM použitých procesorů ATMega.

Wiring je open source programovací platforma pro MCU. Skládá se z programovacího jazyka, integrovaného vývojového prostředí (IDE) a jednočipového MCU. Wiring umožňuje vyvíjet multiplatformní software pro několik architektur MCU [25].

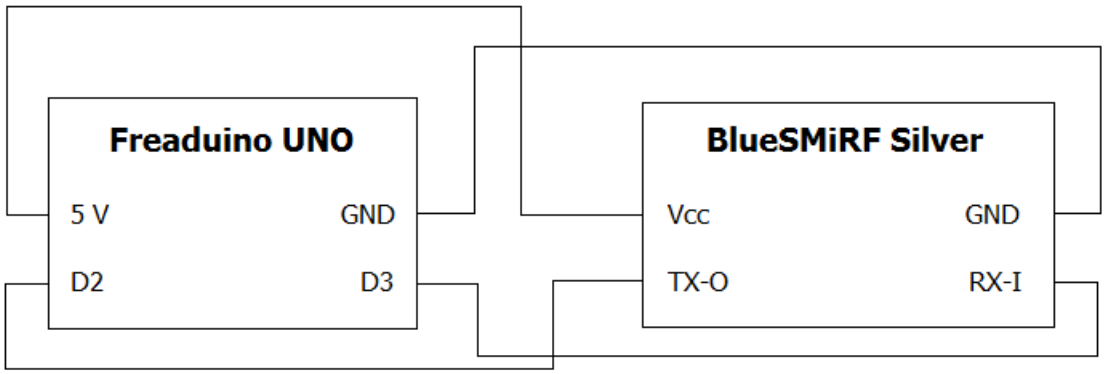
Wiring IDE je multiplatformní software napsaný v Javě a vycházející z Processing IDE. Vývoj probíhá v C/C++ a IDE obsahuje knihovnu „Wiring“ pro usnadnění běžných vstupně výstupních operací.

Psaní programu ve Wiring je odlišné od standardního C/C++. K napsání validního kódu je potřeba napsat dvě funkce: *setup()* a *loop()*. Funkce *setup()* je volána pouze jednou na začátku programu a je vhodná pro počáteční nastavení. Funkce *loop()* je volána periodicky až do resetu nebo odpojení napájení od MCU. Před samotnou kompilací Wiring IDE zkopíruje kód do souboru s jednoduchou *main()* funkcí a vytvoří tak validní C/C++ kód [26].

Processing je open source programovací jazyk a IDE. Původně vyvinutý pro komunitu umělců a designerů s účelem naučení základů programování v grafickém kontextu.

Programovací jazyk Processing je nadstavba Javy a každý Processing soubor je třída definující *Papplet*. Každá třída se ve výsledném Java kódu (generovaný samotným IDE před komiplací) chová jako vnitřní třída [27].

Pro nastavení modulu RN-42 prostřednictvím pinů modulu BlueSMiRF a vývojového kitu Freaduino UNO bylo použito následující zapojení:



Obr. 12 Zapojení pro nastavení modulu RN-42 (na modulu BlueSMiRF Silver) prostřednictvím vývojového kitu Freaduino Uno [24].

Pro nastavení Bluetooth modulu byl do procesoru kitu Freaduino UNO nahrán následující kód (v tomto kódu je použita knihovna pro softwarový UART) [24]:

```
#include <SoftwareSerial.h>

int bluetoothTx = 2;  // TX-O pin modulu BlueSMiRF, připojen k pinu D2 kitu Freaduino UNO
int bluetoothRx = 3;  // TX-O pin modulu BlueSMiRF, připojen k pinu D3 kitu Freaduino UNO

SoftwareSerial bluetooth(bluetoothTx, bluetoothRx);

void setup()
{
  Serial.begin(9600);             // Začni sériovou komunikaci při rychlosti 9600 Bd/s.

  bluetooth.begin(115200);        // Defaultní rychlost modulu je 115200 Bd/s, je tedy nutné začít komunikaci s
                                  // touto rychlostí.
  bluetooth.print("$");
  bluetooth.print("$");
  bluetooth.print("$");           // Vstup do módu nastavení.
  delay(100);                     // Krátká prodleva pro poslání odpovědi ze strany BlueSMiRF modulu.
  bluetooth.println("U,9600,N");  // Dočasně změň rychlost na 9600 Bd/s, žádná parita. 115200 může být příliš
                                  // rychlé pro softwarový UART.
  bluetooth.begin(9600);          // Začni komunikovat přes softwarový UART rychlostí 9600 Bd/s.
}

void loop()
{
  if(bluetooth.available())       // Pokud byla přijata data ze strany Bluetooth modulu
  {
    // Odešli tyto data na sériový port počítače
    Serial.print((char)bluetooth.read());
  }
  if(Serial.available())          // pokud byla přijata data ze sériového portu počítače
  {
    // Odešli tyto data do Bluetooth modulu
    bluetooth.print((char)Serial.read());
  }
  // konec smyčky
}
```

Po nahrání kódu působí kit Freaduino UNO pouze jako prostředník mezi sériovým portem počítače (Arduino používá vlastní sériový port přístupný přímo z Arduino IDE – Serial Monitor). Nyní lze do konzole Serial Monitoru zadávat AT příkazy pro nastavení Bluetooth modulu. Modul RN-42 každý správně formulovaný přijatý příkaz potvrdí odesláním řetězce „AOK“. Současné nastavení

modulu lze vypsat posláním příkazu „D“ pro základní nastavení, nebo „E“ pro rozšířená nastavení. Každý příkaz (kromě příkazu „$$$“, kterým se modul přepne do módu nastavení) musí být zakončen znakem EOL (end of line – nový řádek). Pro nastavení jména modulu, pin kódu a rychlosti přenosu dat je potřeba zadat následující příkazy:

```
$$$              // přejdi do módu nastavení
SN,Kanarek9999   // změň jméno modulu na Kanarek9999
SP,9999          // nastav bezpečnostní pin na 9999
SU,96            // nastav rychlost na 9600 Bd/s
R,1              // restartuj modul pro projevení změn nastavení
```

2.4.2 Programování MCU

Pro programování MCU M430G2553 od společnosti Texas Instruments je k dispozici několik variant. V této kapitole jsou krátce představeny Code Composer Studio IDE a Energia IDE, dvě vývojová prostředí, která je možné použít pro vývoj firmware pro MCU M430G2553. Dále je blíže představen MSP-EXP430G2 LaunchPad, použitý pro nahrání firmware do paměti MCU a popsáno zapojení LaunchPadu a MCU pro programování in-circuit (programování MCU mimo patici LaunchPadu).

2.4.2.1 Vývojová prostředí a firmware

Jednou z možností je použití Code Composer Studio (CCS), oficiálního IDE od Texas Instruments. CCS je založeno na platformě Eclipse. Použití CCS je podmíněno licencí a pro její získání je nutné se zaregistrovat na stránkách Texas Instruments. TI nabízí volnou licenci, která ale omezuje velikost binárního souboru na 32 kB. To je velikost, která je však z praxe dostatečná pro většinu menších a středně velkých projektů (bez použití spousty velkých knihoven), více než dostačující pro projekt *Kanárci*. Vývoj probíhá v programovacím jazyce C a před použitím periferií a IO pinů je nutné je manuálně nakonfigurovat.

Další možností je použití IDE Energia. Energia je fork Arduino IDE a obsahuje knihovny potřebné

pro nahrání a spuštění zkompilovaného kódu do MCU z rodiny MSP430. Programování probíhá stejným způsobem jako pro vývojový kit Freaduino UNO. Programovací jazyk je C/C++ a používá se framework Wiring.

Pro vývoj firmware *Kanárka* bylo zvoleno IDE Energia. Následuje kód nahraného firmware:

```
// definice pinů MCU
int dustPin          = A3;        // vstupní pin AD převodníku
int ledPower         = RED_LED;   // výstupní pin MCU, při logické 1 je zahájeno měření senzoru
int lightDiodePin0   = P2_0;      // indikační LED dioda
int lightDiodePin1   = P2_1;      // indikační LED dioda

// definice konstant
int delayTime0  = 280;
int delayTime1  = 40;
int offTime     = 1000;

// definice proměnných
int dustVal     = 0;   // hodnota přečtená ze senzoru a převedená AD převodníkem
int i           = 0;   // proměnná pro sledování stavu smyčky loop
int j           = 0;   // proměnná pro sledování počtu měření
int k           = 0;   // proměnná pro sledování měřícího cyklu
int dustSum     = 0;   // suma hodnot přečtených ze senzoru a převedených AD převodníkem
int dustAvg     = 0;   // průměrná hodnota měření za dobu jednoho měřícího cyklu

void setup() {

  Serial.begin(9600);
  Serial.println("Kanarek project");
  Serial.flush();

  pinMode(ledPower,        OUTPUT);
  pinMode(lightDiodePin0,  OUTPUT);
  pinMode(lightDiodePin1,  OUTPUT);
  delay(offTime);
}

void loop() {

  Serial.flush();
  i = ~i;
  j = j + 1;

  // Přečtení hodnoty ze senzoru a její převedení do digitální podoby
  digitalWrite(ledPower,        LOW);   // zapne napájení infračervené LED senzoru
  delayMicroseconds(delayTime0);
  dustVal       = analogRead(dustPin);  // přečte hodnotu ze senzoru a převede do digitální podoby
  delayMicroseconds(delayTime1);
```

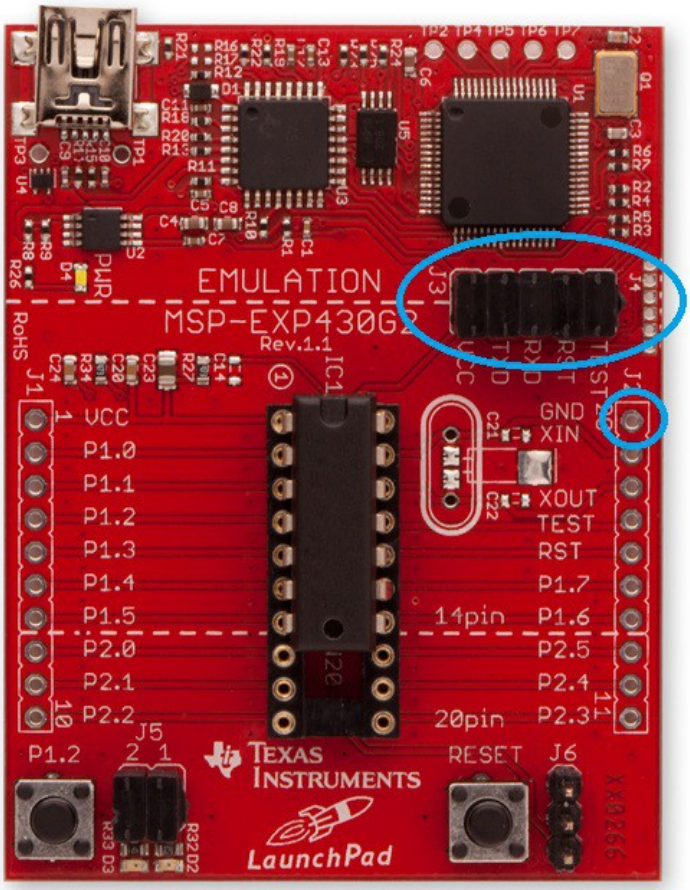
```
  digitalWrite(ledPower,          HIGH);  // vypne napájení infračervené LED senzoru
  dustSum       = dustSum + dustVal;    // přičte přečtenou hodnotu k sumě měřícího cyklu

  // Odeslání výsledku měření rozhraním UART
  if (j == 15)
  {
    // Odesílá se průměr za 15 měření (15 sekund měřící cyklus)
    dustAvg       = dustSum / 15;         // výpočet průměrné hodnoty
    Serial.print(dustAvg);                // odeslání výsledku rozhraním UART
    j             = 0;                    // vynulování počtu měření
    dustSum       = 0;                    // vynulování sumy přečtených hodnot
    k             = 1;                    // indikace odeslání dat
  }

  // Kontroler indikačních LED
  digitalWrite(lightDiodePin0,     i);      // blikání první diody, změna stavu při každém průchodu smyčkou
  digitalWrite(lightDiodePin1,     LOW);   // zhasnutí LED indikující odeslání dat
  if (k == 1)
  {
    // pokud je indikační proměnná rovna 1 (značí odeslání dat)
    digitalWrite(lightDiodePin1,   HIGH);  // rozsviť indikační LED
    k      = 0;                           // vynuluj indikační proměnnou
  }

  // Počkat 1 vteřinu před dalším měřením
  delay(offTime);

  return;
}
```

Ve funkci *setup()* je zahájena sériová komunikace rychlostí 9600 Bd/s. Dále jsou nastaveny operační režimy výstupních pinů.

Smyčka *loop()* nejprve počká na případně nedokončenou odchozí komunikaci na sériové lince. Poté neguje stavovou proměnnou *i* a inkrementuje čítač měření *j*. V každém průběhu smyčky poté následuje přečtení dat ze senzoru. Tato analogová hodnota je převedena AD převodníkem do digitální podoby. Odesílání dat probíhá v 15ti vteřinových intervalech a odesílá se aritmetický průměr hodnot, které se naměřily každou vteřinu. Jako poslední *loop()* nastaví stav LED diod. První dioda mění stav s každým průchodem a značí, že program probíhá. Druhá LED dioda mění stav na jeden průchod smyčky každých 15 vteřin (pokaždé, když proběhne odesílání dat).

2.4.2.2 Programování MCU prostřednictvím MSP-EXP430G2

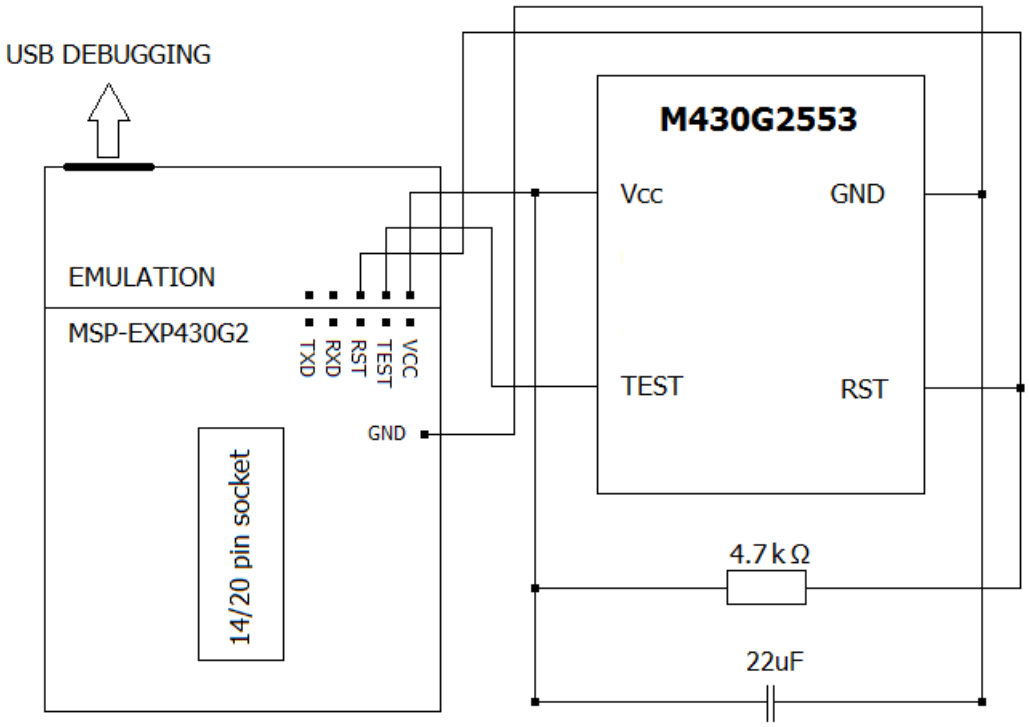
MSP-EXP430G2 LaunchPad je flash programátor a debuggovací nástroj pro rodinu MCU MSP430G2xx. Obsahuje emulátor pro programování a debuggování MCU a 14/20 pinovou DIP (dual in-line package) patici pro MCU [32]. Lze snadno programovat MCU zasazené v patici, nebo MCU připojené v obvodu (pokud by to nešlo, bylo by zapotřebí pro každé nahrání nového firmware přemístit MCU z obvodu zpět do kitu LaunchPad a tím riskovat poškození pinů MCU).



Obr. 13 MSP-EXP430G2 LaunchPad (v patici je umístěno 14ti pinové MCU). Modře jsou označeny jumpery J3 a pin GND mikrokontroleru [32].

K programování MCU mimo patici LaunchPadu je nutné nejprve odstranit všechny jumpery J3 (5

jumperů, které spojují emulační část kitu s částí, ve které je patice s MCU – vpravo nahoře, na obrázku označené větším modrým oválem). Následně je nutné propojit piny TEST, RST a VCC (piny v emulační oblasti LauchPadu, před odstraněním jumperů byly propojeny s piny v oblasti LaunchPadu s MCU) a pin GND (na obrázku označen malým modrým oválem) s piny TEST, RST, VCC a GND na cílovém MCU, které je umístěné v obvodu. Následující schéma blíže zobrazuje popsané propojení.



Obr. 14 Zapojení pro programování MCU M430G2xx prostřednictvím MSP-EXP430G2 LaunchPad mimo patici LaunchPadu

3 Návrh mobilní aplikace

V této kapitole je popsán způsob vývoje mobilní aplikace pro projekt *Kanárci*. Jsou uvedeny požadavky na funkce a vzhled aplikace. Je představen použitý software a knihovny. Dále je popsána struktura aplikace a jednotlivé její části z hlediska uživatelského rozhraní (UI). Jsou popsány protokoly pro komunikaci s *Kanárkem* a pro komunikaci se servery.

3.1 Funkční požadavky

Mobilní aplikace *Kanárci* vychází z již existující aplikace pro platformu iOS. Jakožto verze stejné aplikace pro platformu Android má tedy stejné požadavky na funkce jako původní aplikace pro iOS. Následuje seznam hlavních funkčních požadavků, které aplikace *Kanárci* splňuje.

Zobrazení mapy s vyznačenými pozicemi stanic ČHMÚ a aktivních Kanárků

Aplikace zobrazuje pozice stanic ČHMÚ a dalších aktivních *Kanárků*. Informace o stanicích ČHMÚ a aktivních *Kanárků* jsou ve formě JavaScript Object Notation objektu (JSON objekt) staženy z korespondujících adres a aplikace je parsuje. Na mapě poté zobrazuje pozice stanic i *Kanárků* (stanice ČHMÚ je na mapě zobrazena s jinou ikonou než pozice *Kanárka*) s různě barevnými ikonami podle stupně znečištění, hlášeného danou stanicí či *Kanárkem*. Po kliknutí na ikonu stanice nebo *Kanárka* aplikace zobrazí dialogové okno s podrobnostmi o zvolené stanici, respektive *Kanárkovi*. Dialogové okno obsahuje informace o GPS pozici a naměřené hodnoty.

JavaScript Object Notation (**JSON**) je otevřený standard, který používá člověkem čitelný text pro odesílání datových objektů ve formě *attribute-data* párů. Je využíván hlavně při komunikaci mezi serverey a webovými aplikacemi a tvoří tak alternativu ke standardu XML [33].

Attribute-data pár (*name-value* pár, *key-value* pár) je fundamentální reprezentace dat, tvořící páry, které se skládají z názvu atributu a jeho hodnoty [34].

Příjem dat z Kanárka prostřednictvím Bluetooth a odesílání dat na server

Aplikace pomocí technologie Bluetooth přijímá data odeslaná z *Kanárka*. K tomu účelu je implementován komunikační protokol, který se stará o navázání komunikace se zvoleným *Kanárkem* a periodický příjem dat. Tato data jsou společně s GPS pozicí, na které byla data naměřena, aplikací vizualizována a odeslána na server, který agreguje data o aktivních *Kanárcích*. V rámci komunikačního protokolu Bluetooth také aplikace nabízí jednoduchou správu Bluetooth, která obsahuje vyhledávání dostupných zařízení. Po zvolení *Kanárka* je zařízení uloženo do paměti aplikace a pokud je to nutné (telefon komunikuje se zařízením poprvé), proběhne párování se zařízením. Pro komunikaci pomocí Bluetooth je nutné, aby bylo zvolené zařízení (*Kanárek*), se kterým aplikace komunikuje.

Vizualizace dat

Aplikace vizualizuje naměřená data v reálném čase, kdy zobrazuje poslední naměřenou hodnotu spolu s přepočítaným stupněm znečištění a indikuje stav komunikace společně s odpočtem do příjmu další naměřené hodnoty. Dále aplikace vizualizuje historii měření společně s GPS pozicí měření.

3.2 Vývojové prostředí

V této kapitole je představen software použitý k vývoji mobilní aplikace. Jsou uvedeny použité knihovny, požadavky na cílové zařízení a parametry zařízení, na kterém proběhlo testování.

3.2.1 Použitý software

Java Development Kit (JDK)

Protože Eclipse IDE je napsané primárně v Javě i samotný vývoj aplikace probíhá v Javě, je nutné mít nainstalovaný JDK, který obsahuje JRE (Java Runtime Environment) a nástroje pro vývoj,

monitorování a debugování aplikací.

Eclipse IDE

Open source integrované vývojové prostředí, použité pro programování aplikace. Obsahuje bohatý systém pluginů, které umožňují použití Eclipse IDE pro vývoj v mnoha programovacích jazycích, pro integraci s verzovacími systémy a spoustu dalších funkcí. V této práci bylo využito Eclipse IDE, verze Kepler.

Android Software Development Kit (SDK)

Android SDK obsahuje API (Application Programming Interface) knihovny pro vývoj, testování a debugování aplikací pro OS Android. Před vývojem musí být v Eclipse IDE nastavena cesta k adresáři s Android SDK.

Android Development Tools (ADT) plugin pro Eclipse

Plugin pro Eclipse IDE, který umožňuje snadný vývoj Android aplikací v prostředí Eclipse. Dále nabízí možnost snadného vývoje uživatelského rozhraní pro aplikaci pomocí grafického nebo XML editoru. Instalace ADT probíhá přímo v Eclipse IDE pomocí možnosti *Help → Install New Software*. Podrobné instrukce jsou dostupné na stránkách Android Developers [35].

3.2.2 Použité knihovny

Při vývoji aplikace byly kromě standardních knihoven použity i dvě externí knihovny.

Knihovna **Google Play Services** – její instalace probíhá pomocí SDK Manageru, který je v Eclipse IDE dostupný po správném nastavení Android SDK. Podrobné instrukce k instalaci Google Play Services jsou dostupné na stránkách Android Developers [36].

Knihovna Google Play Services je použita, protože obsahuje Google Maps Android API v2. Google Maps API je v aplikaci použito k implementaci mapy s vyznačenými pozicemi stanic ČHMÚ a *Kanárků*.

Před použitím Google Maps Android API v2 je dále nutné vygenerovat a v aplikaci použít API klíč.

Podrobné instrukce k instalaci a nastavení Google Maps Android API v2 jsou dostupné na stránkách Google Developers [37].

Knihovna **GraphView** – knihovna pro vykreslování flexibilních grafů. V Androidu nelze využít standardních grafických nástrojů AWT a Swing, které nabízí Java SE. Pro kreslení grafů bylo tedy nutné využít externí knihovnu GraphView. GraphView je open source knihovna pro Android, která umožňuje kreslení spojnicových a sloupcových grafů [41].

3.2.2.1 Použití knihoven v Eclipse IDE

Pro importování externí knihovny do projektu v Eclipse IDE existují dvě možnosti.

První možností je zbuildovat projekt, který chceme použít jako knihovnu. Nejprve je nutné zajistit, že se projekt bude buildovat jako knihovna. Toho se v Eclipse docílí tak, že se na něj klikne pravým tlačítkem a zvolí se možnost „Properties“ (možnosti). V levém panelu se zvolí pole „Android“ (pro buildování knihovny pro Android je nutné mít nainstalované ADT a knihovní projekt je nutné založit jako Android Library Project). Na obrazovce, která se objeví je nutné zaškrtnout pole „Is Library“ (je knihovnou). Dalším krokem je v Eclipce zvolit File → Export → Java → JAR File. Na další obrazovce je nutné zvolit knihovní projekt a umístění zbuildované knihovny (soubor JAR – Java Archive). Tento soubor lze pak jako knihovnu přidat do cílového projektu do složky /lib.

Druhou možností je kliknout pravým tlačítkem na cílový projekt, zvolit „Properties“ a v levém panelu „Android“. V sekci „Libraries“ pak tlačítkem „Add“ přidat knihovny. Tento postup je výhodnější, protože umožňuje snadno dělat změny v knihovních projektech. První uvedený postup vyžaduje při každé změně v knihovním projektu nový build a přepsání starého JAR souboru.

Build je proces, při kterém se aplikace připravuje ke zveřejnění (release). Tento proces zahrnuje kompilaci, testování a balíčkování. Po zbuldování projektu v Eclipse IDE je cílový soubor (JAR knihovna, APK soubor atd.) umístěn v projektu ve složce /bin.

3.2.3 Minimální požadavky

Minimální požadavky aplikace na verzi OS Android jsou dány v kódu použitými funkcemi a verzemi API, kterých jsou tyto funkce součástí. APK (Application package file) s aplikací s minimálním požadavkem na verzi systému Android 4.1.x (API verze 16) tak nepůjde nainstalovat na starší verze systému (například Android 4.0.3 (API verze 15)). Minimální požadavek na verzi systému Android je specifikován v souboru *AndroidManifest.xml*, který je součástí každého Android projektu.

Minimální verze systému Android pro nainstalování aplikace *Kanárci* je Android 4.2.x Jelly Bean (API verze 17). Podle hodnot z tabulky 4 je tak aplikace určena pro 32,3 % telefonů se systémem Android. S prodejem telefonů s aktuálnější verzí systému Android se ale toto číslo zákonitě s časem zvyšuje.

Aplikace vyžaduje následující oprávnění:

- android.permission.INTERNET
- kanarci.permission.MAP_RECEIVE
- android.permission.WRITE_EXTERNAL_STORAGE
- com.google.android.providers.gsf.permission.READ_GSERVICES
- android.permission.ACCESS_COARSE_LOCATION
- android.permission.ACCESS_FINE_LOCATION
- android.permission.ACCESS_NETWORK_STATE
- android.permission.BLUETOOTH
- android.permission.BLUETOOTH_ADMIN

Celá aplikace byla průběžně testována na telefonu HTC One X se systémem Android 4.2.2 Jelly bean (API verze 17). Na stejném zařízení pak proběhlo i závěrečné testování aplikace.

3.3 Uživatelské rozhraní

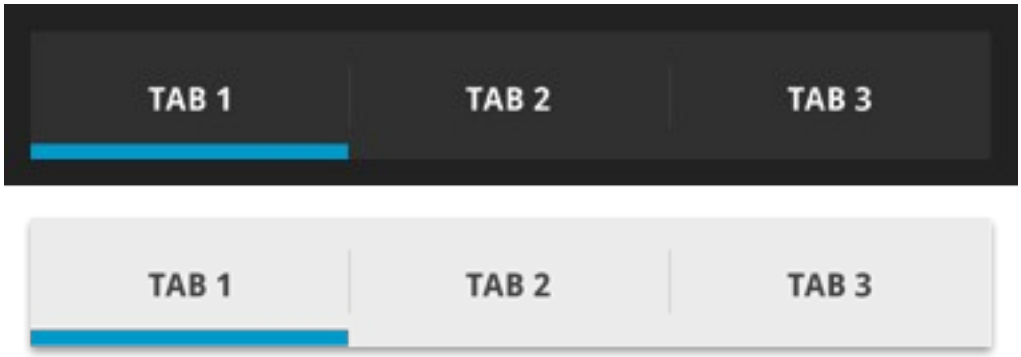
V této kapitole je popsán návrh a realizace uživatelského rozhraní aplikace *Kanárci*. Je popsána architektura aplikace z pohledu UI a představeny její jednotlivé komponenty.

3.3.1 Použité pojmy

V této kapitole jsou krátce vysvětleny pojmy a technické termíny, použité v následující kapitole, která popisuje návrh uživatelského rozhraní.

Tabs (**Taby**) jsou součástí takzvané top-level úrovně aplikace. Top-level úroveň je to první, co uživatel na aplikaci vidí a měl by tedy uživateli poskytnout informace o hlavních funkcích celé aplikace. U většiny moderních aplikací se top-view úroveň skládá z několika hlavních obrazovek, mezi kterými může uživatel listovat. Každá obrazovka pak reprezentuje jednu z hlavních funkcí celé aplikace [38].

Taby jsou v podstatě pevná tlačítka v horní části obrazovky. Jeden z Tabů je vždy aktivní, to znamená, že jedna z hlavních obrazovek je vždy viditelná. Speciálním případem Tabů (použitým v aplikaci *Kanárci*) jsou takzvané **Fixed tabs** (**Pevné taby**). Vyznačují se tím, že jsou vždy uživatelem viditelné (nikdy z obrazovky v rámci dané Aktivity nezmizí). Dále neumožňují skrolování (na rozdíl od Scrollable tabs).



Obr. 15 Příklad Pevných tabů ve dvou barevných schématech [38].

Action bar (**Lišta aplikace**) je na rozdíl od Tabů viditelná v kontextu celé aplikace, její obsah ale může každá Aktivita či Fragment měnit. Opticky je umístěna v horní části obrazovky (pod stavovou lištou a nad Taby). Z pohledu aplikace jsou ale Taby součástí Lišty aplikace. Její hlavní funkce jsou

následující:

- Určuje umístění v rámci aplikace.

- Zobrazuje ikonu aplikace

- Poskytuje místo pro dodatečná tlačítka

- Poskytuje možnost otevřít Overflow menu, které může obsahovat další tlačítka, která se nevešla přímo na lištu. Overflow menu standardně obsahuje tlačítko pro otevření menu nastavení a další tlačítka, společná pro všechny Aktivity a Fragmenty aplikace.

Obr. 16 Příklad Lišty aplikace. Tato lišta obsahuje (1) ikonu aplikace, která může sloužit i pro navigaci zpět v aplikaci, (2) dvě tlačítka na liště a (3) Overflow menu [39].

Fragment je vždy součástí Aktivity. Každý Fragment má vlastní layout i vlastní logiku. Aktivita může obsahovat několik Fragmentů (mezi kterými lze navigovat například pomocí Tabů). To umožňuje integrovat bohaté UI do jediné Aktivity. Fragmenty mají svůj vlastní životní cyklus, který je velice podobný životnímu cyklu Aktivity. Životní cyklus mateřské Aktivity přímo ovlivňuje životní cyklus jejího Fragmentu. Fragmenty však lze přidávat, měnit nebo odebírat z mateřské Aktivity i v průběhu Aktivity [40].

Jedeno z hlavních využití Fragmentů je budování takzvaných multi-pane UI. Jedná se o UI, která nabízejí totožnou funkcionalitu i vzhled, ale dokáží se přizpůsobit fyzické velikosti displaye, na kterém jsou zobrazeny. Pokud je obrazovka dostatečně velká a Aktivita se skládá ze dvou Fragmentů, jsou zobrazeny oba dva najednou. Pokud obrazovka nemá dostatečné rozlišení, pak je zobrazen pouze jeden Fragment a druhý se zobrazí až po uživatelské, nebo jiné akci.

3.3.2 Architektura UI

Jako základ UI aplikace *Kanárci* byl zvolen model Tabs – tlačítka v horní části obrazovky, umístěná pod stavovou lištou a Action Bar aplikace. UI hlavní Aktivity aplikace se pak skládá z pěti Fragmentů, které se přepínají pomocí pěti korespondujících Tabů. Každý Fragment má svůj vlastní Layout (definovaný v odděleném XML souboru) navržený tak, aby vyhovoval funkčním požadavkům jednotlivých Fragmentů. Tyto Fragmenty existují v aplikaci v rámci jedné Aktivity.

Aplikace dále obsahuje dvě další Aktivity. První Aktivita realizuje obrazovku nastavení, ve které má uživatel možnost nastavení způsobu synchronizace aplikace s *Kanárkem* (hardwarovým zařízením). Uživatel má možnost výběru mezi synchronizací pomocí Bluetooth, Softwarového modemu (zatím neimplementováno), nebo vypnutím synchronizace.

Druhá Aktivita realizuje výběr *Kanárka*, se kterým se bude aplikace synchronizovat. Tato možnost je nutná pouze při výběru synchronizace pomocí Bluetooth. Tlačítky v Action Bar této Aktivity má uživatel možnost zapnout a vypnout Bluetooth adaptér a vyhledat dostupná Bluetooth zařízení. Aktivita dále zobrazuje aktuálně vybrané zařízení, nebo informaci o tom, že žádné zařízení vybrané není. Pro synchronizaci je třeba vybrat *Kanárka*, se kterým bude aplikace následně komunikovat. Pokud se zařízení s *Kanárkem* spojilo poprvé, tato Aktivita zahájí párovací proces, který se poté odehrává na systémové úrovni.

V následujících kapitolách jsou blíže popsány jednotlivé Fragmenty a jejich UI.

3.3.2.1 Fragment Měření

Tento fragment je hlavním Fragmentem hlavní Aktivity. Je tedy tím prvním, co uživatel vidí. Slouží ke spuštění a vypnutí měření a k vizualizaci výsledků. V průběhu měření zobrazuje poslední změřenou hodnotu formou textu i graficky. Dále zobrazuje textový popisek k poslednímu měření a datum a čas posledního měření. Dále tento Fragment obsahuje tlačítko, kterým se měření zapne nebo vypne (textový obsah tlačítka se mění podle současného stavu aplikace). Pokud je měření vypnuté, zobrazuje Fragment poslední měření, nebo žádná data v případě, že v průběhu životního cyklu aplikace k žádnému měření nedošlo.

Pro zahájení měření při komunikaci s *Kanárkem* prostřednictvím Bluetooth je nutné, aby byl Bluetooth adaptér telefonu zapnutý a aby bylo vybrané zařízení (*Kanárek*) pro komunikaci. Pokud není jedna z podmínek splněna, k zahájení komunikace nedojde.

Naměřené hodnoty se kategorizují podle následující tabulky.

Kvalita ovzduší	PM10 [μg/m^3]
Velmi dobrá	0 - 20
Dobrá	20 - 40
Uspokojivá	40 - 70
Vyhovující	70 - 90
Špatná	90 - 180
Velmi špatná	> 180

Tab. 9 Stupně znečištění ovzduší polétavými částicemi.

Obr. 17

Fragment Měření při nezměřené hodnotě

Obr. 18

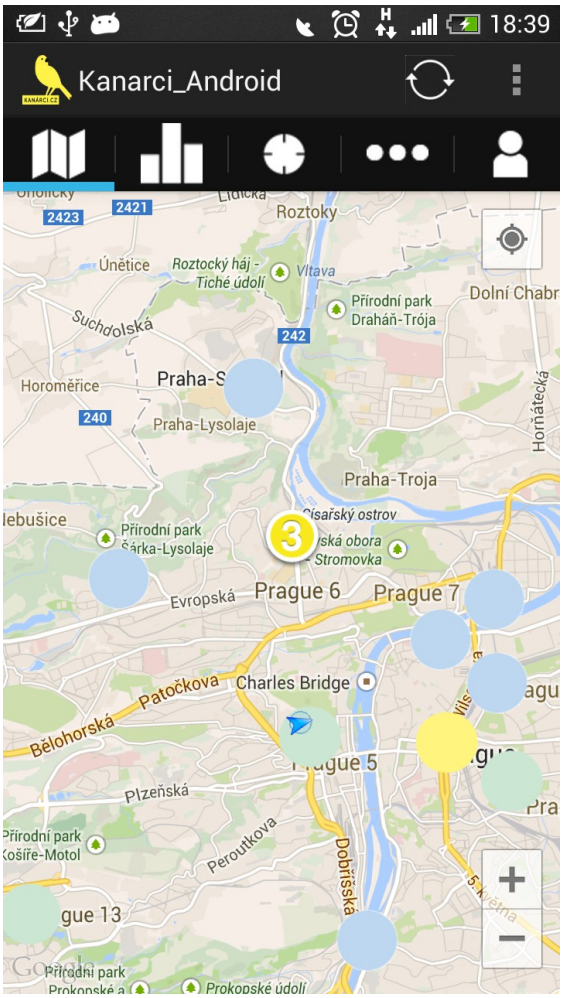
Fragment Měření při naměřené hodnotě

3.3.2.2 Fragment Mapa

Tento Fragment obsahuje objekt MapFragment. Ten funguje jako funkční obal objektu GoogleMap, který je součástí Google Maps API a zprostředkovává možnost využití Google map v aplikaci. Uživatelské rozhraní mapy se stává z lokačního tlačítka (pravý horní roh), které po stisknutí prostřednictvím GPS určí polohu zařízení a na mapě jí znázorní. UI dále obsahuje zoomovací tlačítka v dolním pravém rohu obrazovky.

Lišta aplikace obsahuje pro tento Fragment jedno přídavné tlačítko. Jeho stisknutím se zahájí stahování dat o *Kanárcích* a stanicích ČHMU. Jejich pozice jsou poté vyznačeny na mapě a stav znečištění, které hlásí, je indikován barvou ikony, kterou na mapě mají (barvy odpovídají tabulce 9). Jiným stylem ikony jsou také označeny stanice ČHMÚ a *Kanárků*. Ikona Stanice ČHMÚ je barevné kolečko bez výplně, ikona *Kanárka* je barevné kolečko s bílým okrajem a číslem uvnitř, které značí stupeň znečištění (také podle tabulky 9).

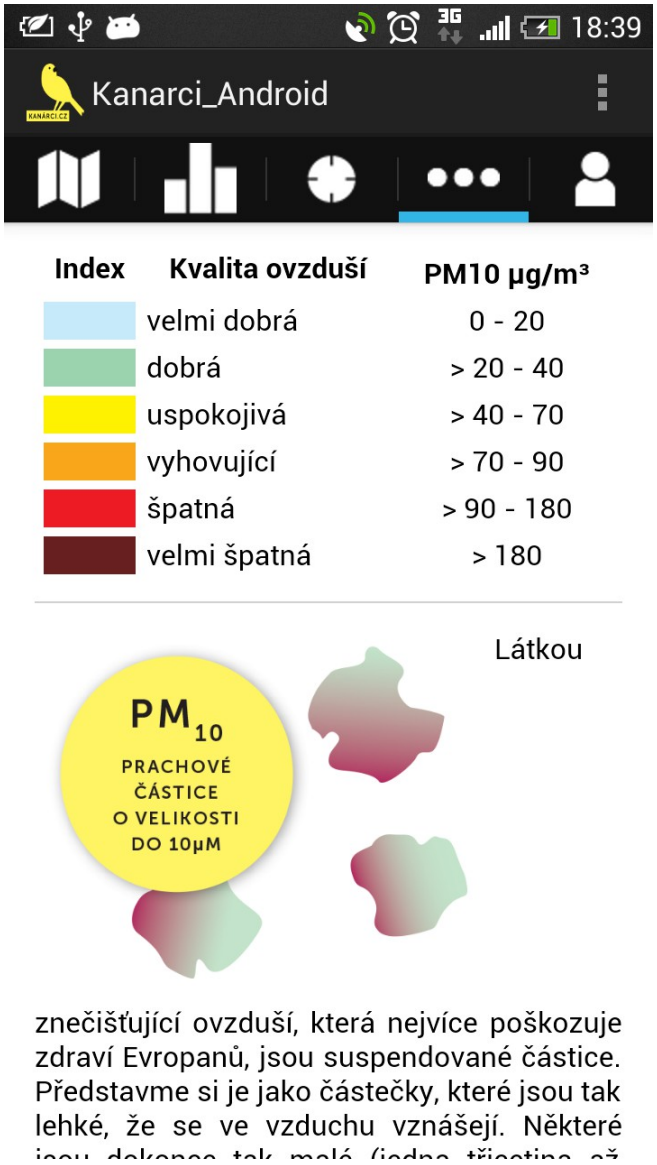
Po kliknutí na ikonu stanice ČHMÚ nebo *Kanárka* se uživateli zobrazí dialogové okno obsahující podrobné informace o dané stanici nebo *Kanárkovi*. Tyto informace zahrnují stavy sledovaných znečišťujících prvků (v případě *Kanárků* se jedná pouze o PM10), čas měření a lokalitu.



Obr. 19 Fragment Mapa zobrazující několik stanic ČHMÚ v Praze a jednoho *Kanárka*

3.3.2.3 Fragment Legenda

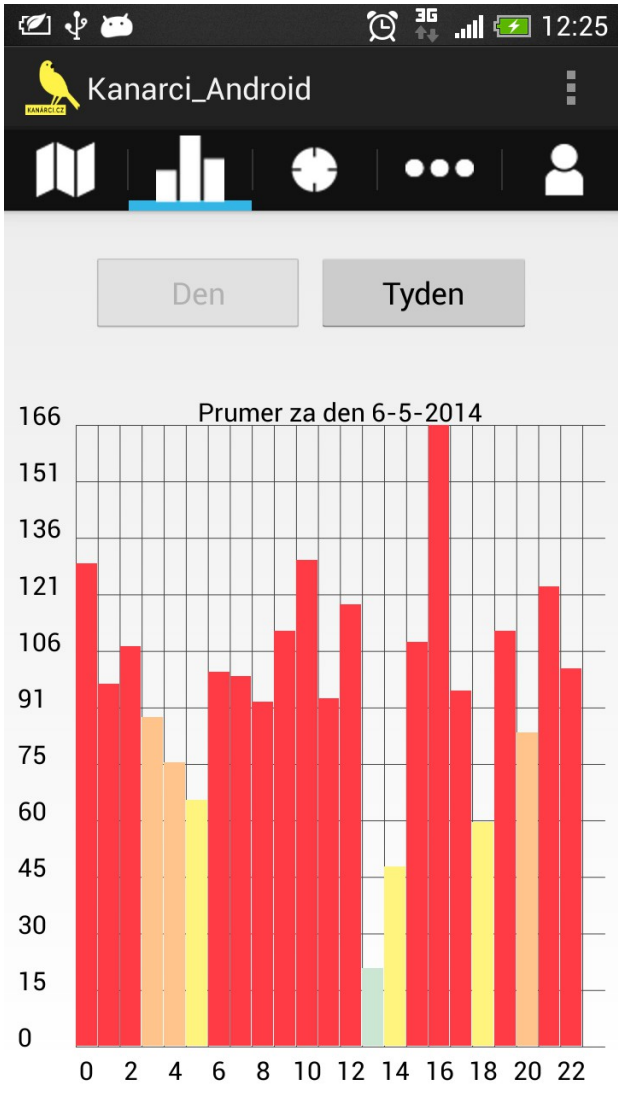
Tento Fragment je pouze informativní a postrádá jakoukoliv funkčnost nebo prvky UI. Jeho podstatou je poskytnout uživateli základní a důležité informace o polétavém znečištění, jeho měření a jeho mezních hodnotách. Fragment obsahuje objekt WebView, který zobrazuje webovou stránku, která byla speciálně navržena pro zobrazení mobilními zařízeními. Toto řešení má dvě výhody. Redukuje kód aplikace a také ho činí nezávislým na jakýchkoliv změnách těchto informací.



Obr. 20 Fragment Legenda zobrazující webovou stránku pomocí objektu WebView

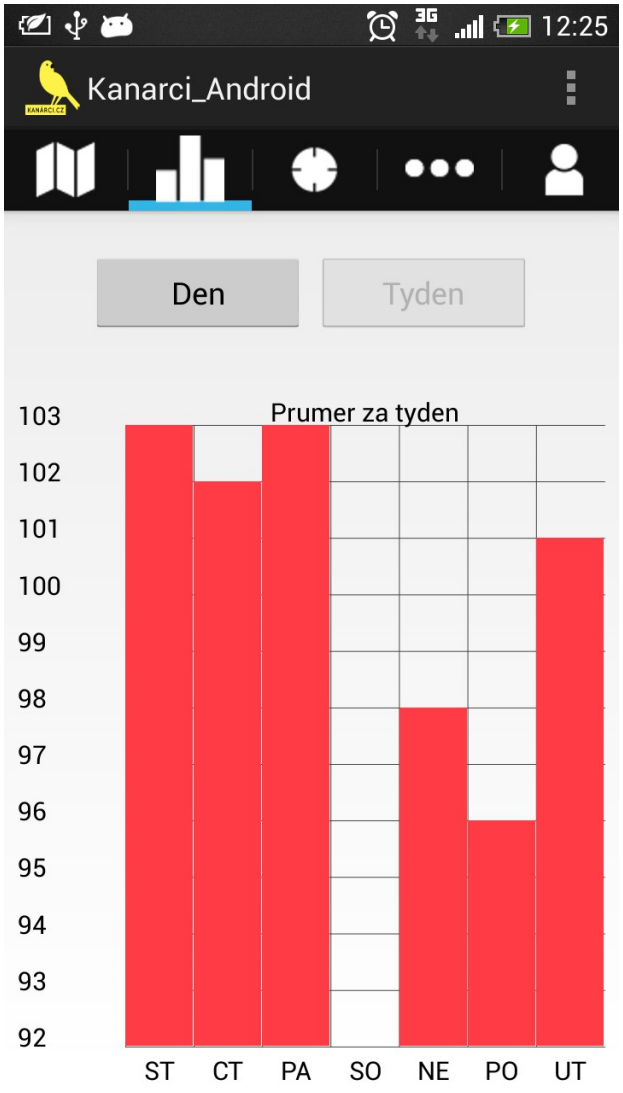
3.3.2.4 Fragment Statistiky

Fragment Statistiky zobrazuje průměrné hodnoty naměřené během předchozího dne (zobrazené hodnoty jsou průměry hodinového měření) nebo předchozího týdne (zobrazené hodnoty jsou průměry denních měření). Uživatel má možnost přepínat mezi grafy pomocí dvou tlačítek v horní části obrazovky. Grafy jsou vykresleny pomocí knihovny GraphView. Graf se zabarvuje podle hodnoty znečištění (barvy a limitní hodnoty odpovídají tabulce 9).



Obr. 21

Fragment Statistiky zobrazující denní výpis



Obr. 22

Fragment Statistiky zobrazující týdenní výpis

3.4 Implementace funkcí aplikace

V této kapitole je popsán způsob implementace stěžejních funkcí mobilní aplikace. Jsou uvedeny použité postupy a knihovny. Jsou popsány třídy, které funkcionalitu zajišťují. Vývoj mobilní aplikace probíhal v Javě. Jsou zmíněná použitá Java a Android API.

3.4.1 Databáze

Databáze je v aplikaci využita k uchování informací o jednotlivých měřeních. V systému Android je vestavěn databázový systém SQLite. Používání databáze SQLite v Androidu nevyžaduje žádné počáteční nastavení nebo administraci. Je potřeba pouze definovat databázové příkazy, o zbytek se stará samotná platforma Android.

V aplikaci *Kanárci* je k databázi přistupováno pomocí třídy **Database**. Tato třída zajišťuje životní cyklus databáze SQLite vytvořené aplikací. Ve třídě Database jsou definovány všechny databázové příkazy. K databázi přistupuje pomocí podtypu *SQLiteOpenHelper* (API *android.database.sqlite*), které zajišťuje otevírání a uzavírání SQLite databáze.

Třída Database je implementována ve stylu návrhového vzoru Singleton. To zajišťuje, že v rámci celé aplikace bude vždy existovat pouze jediná instance třídy Database. To je nutné, protože aplikace přistupuje do databáze z různých míst. Pokud by v každém z těchto míst byla vytvořena nová instance třídy Database, která by přistupovala ke stejné SQLite databázi, docházelo by k několika současným přístupům a to by vyvolalo výjimku.

Třída Database implementuje CRUD (**C**reate, **R**ead, **U**pdate, **D**elete) operace pro práci s databází. Funkce *create(Measurement m)* ukládá data do databáze. Data jsou této funkci předána v podobě instance třídy **Measurement**. Třída Measurement obsahuje informace o měření. Tyto informace zahrnují datum a čas měření, GPS souřadnice měření (pouze pokud byly v době měření dostupné) a hodnotu měření. Funkce *read(Calendar c)* čte data z databáze. Je implementována tak, že přečte vždy všechny záznamy ze dne, který je určený jejím parametrem (instance třídy *Calendar* (API *java.util*)). Funkce *read* vrací objekt třídy Cursor (API *android.database*).

Databáze obsahuje jedinou tabulku s následující strukturou:

_ID (primary key autoincrement), LAT (double), LON (double), HOUR (integer), MINUTE (integer), SECOND (integer), DAY (integer), MONTH (integer), YEAR (integer), VALUE (double)

Pole *_ID* je primární klíč záznamu v tabulce, pole *LAT* a *LON* jsou GPS souřadnice měření, pole *VALUE* je hodnota naměřeného znečištění. Zbylá pole určují datum a čas měření.

SQLite je open source databázový systém. V současnosti je SQLite nejrozšířenějším databázovým systémem na světě. Největší využití nachází SQLite ve vestavěných systémech včetně mobilních zařízení. Kromě systém Android využívá databázový systém SQLite i konkurenční iOS, Windows Phone i Blackberry. SQLite čte a zapisuje přímo z disku a celá databáze včetně všech jejích tabulek je uložena v jediném souboru, který je multiplatformní.

Její hlavní výhodou s ohledem na využití v mobilních zařízeních je její velikost. Kompletní knihovna SQLite se všemi funkcemi může zabírat méně než 500 kB v závislosti na kompilátoru a na cílové architektuře [42].

Následující kód demonstruje konstrukci třídy Database podle návrhového vzoru Singleton. Instance třídy Database se získá voláním statické funkce *Database.getInstance()*.

```
public class Database {
	private static Database instance;
	private Database() {
		// privátní konstruktor
	}
	public static Database getInstance() {
		if (instance == null) {
			instance = new Database();
		}
		return instance;
	}
}
```

3.4.2 Komunikace se serverem

Aplikace komunikuje se servery za účelem získání dat o stanicích ČHMÚ a dalších *Kanárcích*. Tato data jsou poté zobrazena prostřednictvím Fragmentu Mapa.

Komunikace probíhá prostřednictvím protokolu Hypertext Transfer Protocol (HTTP) a programově jí zajišťuje třída **Communicator**. Třída Communicator ctí návrhový vzor Singleton. Veškeré HTTP požadavky musí probíhat mimo vlákno UI a tedy ve svém separátním vlákně. K tomu třída Communicator využívá podtypu *AsyncTask<Params, Progress, Result>* (API *android.os*). Ve třídě jsou definovány dva podtypy *AsyncTask*, každé určené pro komunikaci s jiným serverem. Aplikace získává data o stanicích ČHMÚ z adresy *http://www.vrbka.me/smogalarm/* a data o *Kanárcích* z adresy *http://api.xively.com/v2/feeds?q=kanarek&status=live*. Data na serveru V*rbka* jsou veřejně přístupná, pro přístup k datům na serveru *Xively* je nutné přidat k HTTP požadavku API klíč. Pokud server odpoví stavovým kódem 200 (OK, request was fulfilled [43]) pak celý obsah odpovědi uloží do instance třídy **RawDataPack** ve formě textového řetězce. Dále do tohoto objektu uloží i stavový kód, kterým server odpověděl a nastaví flag *dataReady*, který značí, že data v objektu RawDataPack jsou připravena k použití. Pokud server odpoví jiným stavovým kódem než 200, pak se do objektu RawDataPack neukládá obsah odpovědi, ale pouze stavový kód a pokud při komunikaci došlo k výjimce (možné výjimky, které mohou nastat jsou *ClientProtocolException* a *IOException*), ukládá se do tohoto objektu také textová reprezentace této výjimky.

Data ze serveru, uložená v podobě textového řetězce v objektu RawDataPack, se dále musí parsovat. O to se stará třída **JSONParser**. Stejně jako Communicator je JSONParser Singleton. Obsahuje funkci *getJSONObject(String raw_string)*, která převede textový řetězec do podoby *JSONObject* (API *org.json*). Dále obsahuje funkce *FormatedDataPack parseInputDataVrbka(RawDataPack raw_data)* a *FormatedDataPack parseInputDataXively(RawDatapack raw_data)*. Tyto funkce vytváří instance tříd **FormatedDataPack**. Třída FormatedDataPack obsahuje informaci o datu a času, která byla funkcemi parsována a dále dvě instance seznamů *ArrayList* (API *java.util*) *stations*<***Station***> a *kanarci*<***Kanarek***>. Tyto seznamy obsahují informace o stanicích ČHMÚ nebo *Kanárcích*, které byly rozparsované z objektu RawDataPack. Seznamy obsahují instance tříd Station a Kanarek, které jsou

implementovány tak, aby uchovávaly důležité informace o stanicích ČHMÚ respektive *Kanárcích*, které byly staženy se serverů *Vrbka* a *Xively*.

Instance třídy FormatedDataPack již obsahuje informace, které jsou připravené a aplikací dále použitelné.

Následující ukázka kódu demonstruje konstrukci a spuštění podtypu *AsyncTask<Params, Progress, Result>*.

```
class AsyncTaskDemo extends AsyncTask<Void, Integer, String> {
	@Override
	protected void onPreExecute() {
		// Zde napsaný kód se vykoná před zahájením běhu vlákna.
	}
	@Override
	protected String doInbackground(Void... params) {
		// Zde napsaný kód tvoří tělo vlákna. Funkce musí vracet objekt typu Result.
		publishProgress(progress);
		return result;
	}
	@Override
	protected void onProgressUpdate(Integer... progress) {
		// tato funkce je volána pokaždé, když dojde ke změně parametru Progress. Zde napsaný kód
		// lze využít například k aktualizaci UI.
	}
	@Override
	protected void onPostExecute(String result) {
		// zde napsaný kód se vykoná po ukončení běhu vlákna (poté, co funkce doInBackground vrátí
		// výsledek).
	}
}
new AsyncTaskDemo().execute(/* parametry, v tomto příkladu Void, tedy není třeba žádné udávat */);
```

3.4.3 Zobrazení mapy

K zobrazení mapy ve Fragmentu Mapa je využito knihovny Google Play Services. Samotné zobrazení mapa tedy vyžaduje minimum kódu na straně aplikace a je uskutečněno pouze voláním funkcí Google Maps Android API v2.

Pro přidání markerů na mapu je nejprve nutné stáhnout data o stanicích ČHMÚ a *Kanárcích* ze serverů *Vrbka* a *Xively*. O stažení, parsování a formátování dat se starají třídy Communicator a

JSONParser. Fragment Mapa pak pouze používá výsledné instance třídy FormatedDataPack. S použitím seznamů *stations* a *kanarci* pak přidává markery na mapu. Markerům je přiřazena ikona podle toho, jedná-li se o stanici ČHMÚ, nebo *Kanárka*. Barva ikony je dále řízena stupněm znečištění, které jednotlivé stanice a *Kanárci* hlásí. Přepočet je rozdílný pro stanice ČHMÚ a *Kanárky*.

Stanice ČHMÚ měří více znečišťujících prvků. Konkrétně se jedná o PM10, CO, NO_2, O_3 a SO_2. Hladina znečištění (stupně podle tabulky 9) se pak určuje jako maximum ze stupňů znečištění jednotlivými látkami. Následující tabulka znázorňuje limitní hodnoty pro zařazení naměřených hodnot jednotlivých látek do stupňů znečištění definovaných v tabulce 9.

	PM10 [μg/m^3]	CO [μg/m^3]	NO_2 [μg/m^3]	O_3 [μg/m^3]	SO_2 [μg/m^3]
Velmi dobrá	0 - 20	0 - 1000	0 - 25	0 - 33	0 - 25
Dobrá	20 - 40	1000 - 2000	25 - 50	33 - 65	25 - 50
Uspokojivá	40 - 70	2000 - 4000	50 - 100	65 - 120	50 - 120
Vyhovující	70 - 90	4000 - 10000	100 - 200	120 - 180	120 - 350
Špatná	90 - 180	10000 - 30000	200 - 400	180 - 240	350 - 500
Velmi špatná	> 180	> 30000	> 400	> 240	> 500

Tab. 10 Stupně znečištění pro látky měřené stanicemi ČHMÚ.

Kanárci měří pouze polétavé znečištění PM10. Hodnota ze senzoru SHARP ale musí být před kategorizací přepočítána převodní funkcí. Tuto funkci složil Bc. Jan Remeš, který na projektu *Kanárci* v době vyhotovení této práce také pracoval. Převodní funkce pro senzor SHARP je následující:

$$pm10 = 1.365 \cdot SHARP - 188.088$$

Kliknutí na marker na mapě vyvolá zobrazení dialogového okna s informacemi o dané stanici nebo *Kanárkovi*. Následující fragment kódu znázorňuje definici a registraci rozhraní pro kliknutí na marker.

```
class MyOnMarkerClickListener implements OnMarkerClickListener {
	@Override
	public boolean onMarkerClick(Marker marker) {
		final Dialog dialog = new Dialog(getActivity());
		dialog.setCancelable(true);

		// Zde proběhne nastavení obsahu dialogového okna

		dialog.show();
		return true;
	}
}
private GoogleMap map;
map.setOnMarkerClickListener(new MyOnMarkerClickListener());
```

3.4.4 Komunikační rozhraní Bluetooth

Aplikace používá Bluetooth pro komunikaci s *Kanárkem*, respektive s Bluetooth modulem BlueSMiRF Silver, který *Kanárek* pro komunikaci využívá. Na straně aplikace se o komunikaci stará třída **BluetoothService**. BluetoothService je navržena jako Singleton. Pro zahájení komunikace slouží funkce *synchronized void connect(BluetoothDevice device)*. Parametrem této funkce je instance třídy *BluetoothDevice* (API *android.bluetooth*).

Dále BluetoothService implementuje dvě vlákna. První vlákno *ConnectThread* je spuštěno ve funkci *connect* a jeho úkolem je navázat spojení s Bluetooth zařízením specifikovaným instancí *BluetoothDevice*. Nejprve musí aplikace vytvořit instanci třídy *BluetoothSocket* (API *android.bluetooth*). Toho je docíleno voláním funkce *BluetoothDevice.createRfcommSocketToServiceRecord(UUID uuid)*. Funkce musí být volána na zařízení, ke kterému se aplikace snaží připojit. Tato funkce vrátí RFCOMM (Radio Frequency Communication) *BluetoothSocket*, který je připravený zahájit odchozí komunikaci se vzdáleným zařízením s použitím Session Description Protocol (SDP), specifikovaným parametrem *uuid* [44]. Aplikace používá SPP (Serial Port Profile) *UUID*: 00001101-0000-1000-8000-00805F9B34FB.

Po úspěšném vytvoření instance *BluetoothSocket* se vlákno *ConnectThread* pokusí tímto soketem připojit ke vzdálenému zařízení. Připojení se docílí voláním funkce *BluetoothSocket.connect()* na právě vytvořené instanci třídy *BluetoothSocket*. Je-li připojení úspěšné, rutina tohoto vlákna končí a

je spuštěno druhé vlákno, implementované třídou BluetoothService, *ConnectedThread*.

Vlákno *ConnectedThread* se stará o otevřené spojení se vzdáleným Bluetooth zařízením. Po spuštění vlákna získá instanci *InputStream* (API *java.io*) voláním *BluetoothSocket.getInputStream()* na instanci *BluetoothSocket*, kterou bylo spojení navázáno. V rutině pak vlákno čeká v nekonečné smyčce na obsah *InputStream* a vyčítá ho do pole bytů. Pokud jsou přečtena data, jsou nejprve z tohoto pole, kde jsou uložena ve formě ASCII znaků, převedena na *Integer*. Poté vlákno vytvoří instanci třídy *Calendar*, která reprezentuje čas, kdy byla data přijata a instanci třídy *LatLng* (API *gms.maps.model*). Instance *LatLng* je vytvořena pomocí třídy **LocationService**. Vlákno dále vytvoří novou instanci třídy Measurement a vloží jí do databáze prostřednictvím instance Database. Poslední věcí, kterou vlákno s daty udělá je odeslání instance Measurement do vlákna UI. UI vlákno tato data použije k aktualizaci UI Fragmentu Měření.

Při vytváření instance BluetoothService je konstruktoru předán objekt třídy *Handler* (API *android.os*). Pomocí toho mohou vlákna implementovaná třídou BluetoothService odesílat zprávy vláknu, které *Handler* vlastní (v tomto případě je to vlákno UI). Odesílání zpráv probíhá voláním funkce *Handler.obtainMessage(int what, int arg1, int arg2, Object obj).sendToTarget()*. Tímto voláním lze k odesílané zprávě přiřadit i Objekt (v tomto případě instanci třídy Measurement vytvořenou z přijatých dat). Pro odeslání zprávy, obsahující pouze jeden *Integer* (pro upozornění vlákna UI na dění uvnitř vlákna *ConnectThread* nebo *ConnectedThread*) lze využít volání *Handler.obtainMessage(int what).sendTotarget()*. Obě volání vytváří instanci třídy *Message* (API *android.os*), kterou vlákno, vlastnící použitý *Handler*, může přijmout. Následující fragment kódu demonstruje implementaci instance třídy *Handler*.

```
private static final Handler mHandler = new Handler () {
        @Override
        public void handleMessage(Message msg) {
                switch(msg.what) {
                        // zde implementované chování vlákna po přijetí různých zpráv
                }
        }
};
```

Thread (**Vlákno**) je nezávislé vykonávání sekvence instrukcí, řízené plánovačem procesů (Scheduler) operačního systému. Ve většině operačních systémů jsou vlákna součástí procesů. V rámci jednoho procesu může existovat několik vláken, které sdílejí stejné zdroje, například paměť. Na rozdíl od vláken, procesy navzájem tyto zdroje nesdílejí [45].

V OS Android je vlákno chápáno jako souběžná jednotka vykonávání instrukcí. Vlákno má svůj vlastní zásobník pro volání funkcí, jejich argumenty a lokální proměnné. Každá aplikace má minimálně jedno hlavní vlákno (UI vlákno) [46].

Implementace vlákna je možná dvěma způsoby. Lze vytvořit podtřídu třídy *Thread* (API *java.lang*), nebo rozhraní *Runnable* (API *java.lang*).

3.4.5 Zjištění GPS souřadnic

Zjišťování souřadnic GPS obstarává třída LocationService. LocationService je konstruována jako Singleton. Obsahuje funkce *startLocationService()* a *stopLocationService()*. Třída využívá systémovou službu *LocationManager* (API *android.location*) a rozhraní *LocationListener* (API *android.location*). Při volání funkce *startLocationService()* si aplikace vyžádá notifikace o nové GPS poloze voláním *LocationManager.requestLocationUpdates(String provider, long minTime, float minDistance, LocationListener listener)*. Parametr *provider* specifikuje poskytovatele (provider), který zjišťuje polohu pomocí GPS satelitů. Ve funkci je použit standardní provider *LocationManager.GPS_PROVIDER*. Parametry *minTime* a *minDistance* specifikují minimální časový úsek mezi notifikacemi o nové poloze v milisekundách a minimální vzdálenost mezi notifikacemi v metrech. V aplikaci jsou použity hodnoty 5000 a 10. Parametr *listener* je instance rozhraní *LocationListener*, který zachytí notifikaci o nové poloze a provede svou rutinu. Instanci rozhraní *LocationListener* definuje třída LocationService následujícím způsobem.

```
public class LocationService {
	private double lat, lon;
	private LocationListener listener = new MyLocationListener();
	private class MyLocationListener implements LocationListener {
		@Override
		public void onLocationChanged(Location location) {
			// zachycena notifikace o nové GPS poloze
			lat = location.getLatitude();
			lon = location.getLongitude();
		}
```

```
            @Override
            public void onProviderDisabled(String provider) {

            }
            @Override
            public void onProviderEnabled(String provider) {

            }
            @Override
            public void onStatusChanged(String provider, int status, Bundle extras) {

            }
        }
}
```

3.4.6 Kreslení grafů ve Fragmentu Statistiky

Kreslení grafů ve Fragmentu Statistiky je prováděno pomocí externí knihovny GraphView. Uživatel má na výběr zobrazení grafu za minulý den, nebo za uplynulý týden. V obou případech jsou data získána z databáze.

Pro případ denního grafu jsou získána data pouze za minulý den (pokud je dnes například pondělí 17.5, pak jsou z databáze přečtena všechna měření ze dne neděle 16.5). Algoritmus poté zprůměruje všechna měření podle hodiny dne, kdy byla naměřena a zobrazí je do sloupcového grafu, kde na svislé ose jsou průměry měření a na vodorovné ose hodiny dne. Graf tedy zobrazuje 24 sloupců (pokud jsou data dostupná z každé hodiny předchozího dne).

V případě týdenního grafu jsou měření průměrována v rámci dnů. Algoritmus tedy přečte hodnoty za uplynulých 7 dní a hodnoty zprůměruje. Data jsou pak vynesena do sloupcového grafu, kde na svislé ose jsou průměry měření a na vodorovné ose dny v týdnu. Graf tedy zobrazuje 7 sloupců (pokud jsou data dostupná z každého z posledních 7 dnů). Dny na vodorovné ose jsou seřazeny podle aktuálního dne a graf zobrazuje dny od nejstaršího po nejnovější. Pokud je tedy dnes například čtvrtek, vodorovná osa začíná čtvrtkem z minulého týdne a končí středou (včerejškem).

Pro práci s knihovnou GraphView je dále potřeba vytvořit rozhraní *GraphViewDataInterface* (API *com.jjoe64.graphview*). Toto rozhraní se skládá pouze ze dvou hodnot typu *Double* a příslušných getterů a setterů.

Následující fragment kódu demonstruje práci s knihovnou GraphView.

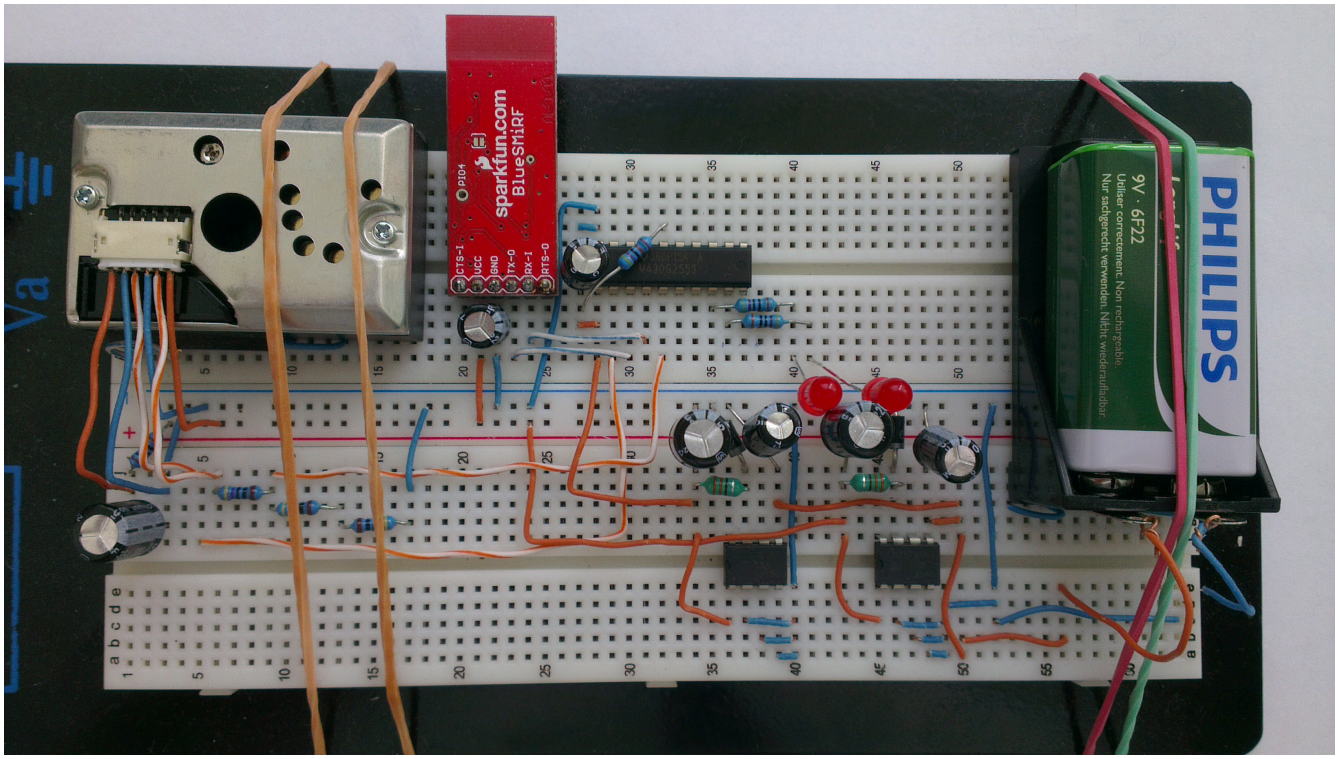
```
GraphViewData[] data = new GraphViewData[24];        // Třída GraphViewData je rozhraním
                                                     // GraphViewDataInterface
for (int i = 0; i < 24; i++) {
        data[i] = new GraphViewData(i, /* zde průměrná hodnota za hodinu i v uplynulém dni */);
}
GraphViewSeries series = new GraphViewSeries("Prumer za uplynuly den", data);
GraphView graphView = new BarGraphView("Prumer za uplynuly den");
graphView.addSeries(series);
graphView.setHorizontalLabels(new String{"0", "1", "2", ... "22", "23"});
LinearLayout layout = (LinearLayout) findViewById(R.id.layout);
layout.addView(graphView);
```

Gettery a **Settery** jsou funkce s veřejným přístupem. Jejich účelem je přistupovat k privátním proměnným a nastavovat jejich hodnotu (Settery) nebo jí číst (Gettery). Následující fragment kódu demonstruje konstrukci třídy s jednou privátní proměnnou, konstruktorem a Getterem a Setterem.

```
public class ClassDemo {
        private int value;
        public ClassDemo(int value) {
                // Konstruktor
                this.value = value;
        }
        public void setValue(int value) {
                // Setter
                this.value = value;
        }
        public int getValue() {
                // Getter
                return value;
        }
}
```

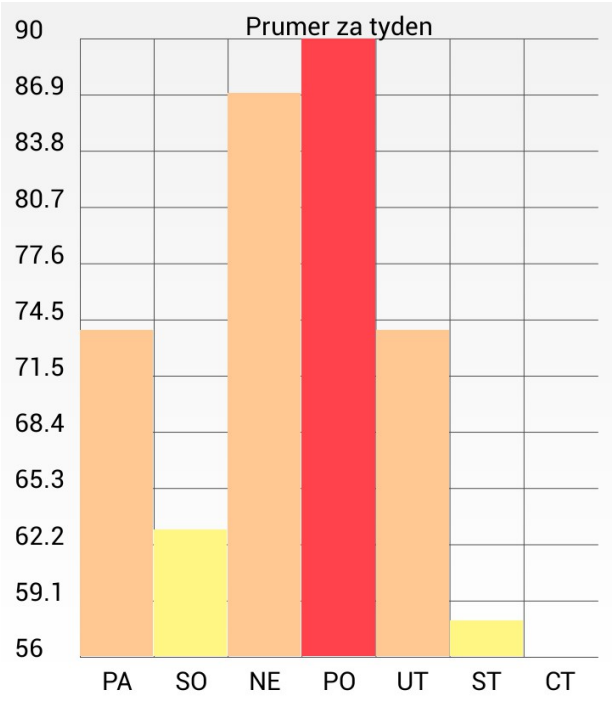
4 Testování

Testování proběhlo v časovém úseku jednoho týdne. Každý den testování bylo měření spuštěno během dne, alespoň v pěti různých hodinách po dobu alespoň čtyř měření (60 sekund). Měření probíhalo na pokoji vysokoškolské koleje.

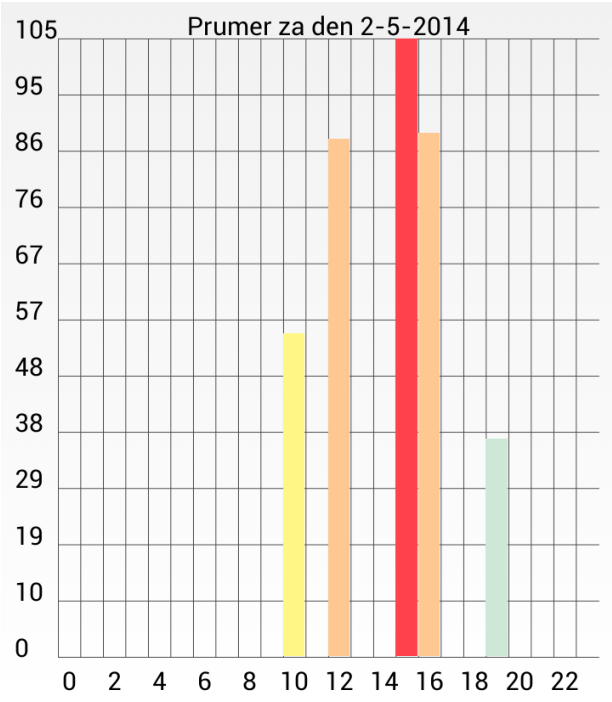


Obr. 23 Sestrojený prototyp mobilního Kanárka na kterém proběhlo testování.

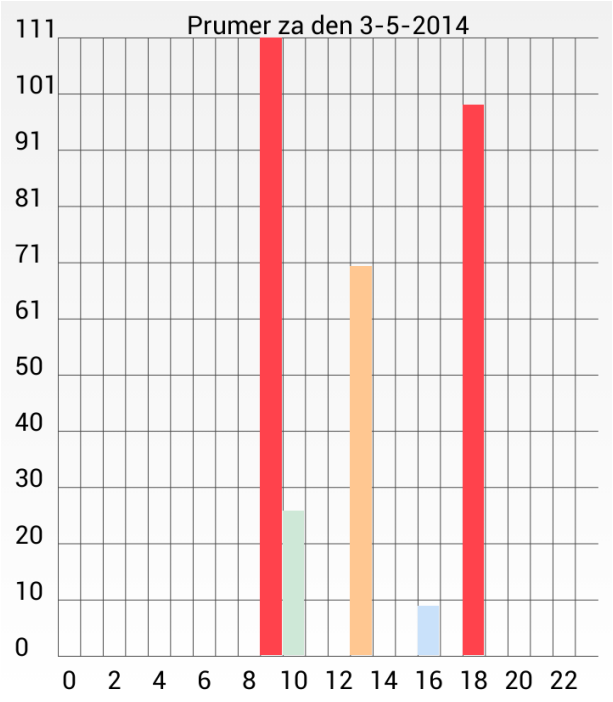
Následující grafy jsou vyňaty z Fragmentu Statistiky. Ukazují statistiku za celý testovací týden i statistiku za každý den testovacího týdnu.



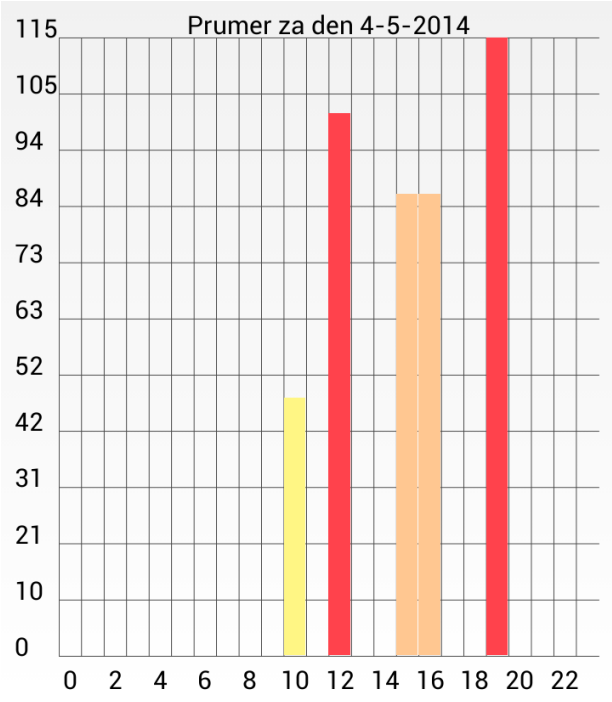
Obr. 24 Statistika testovacího týdne.



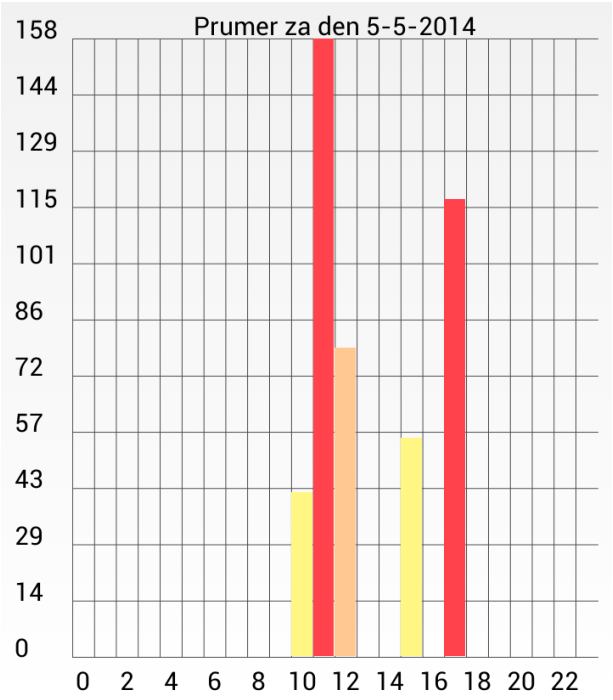
Obr. 25 Statistika testovacího dne 2.5.2014.



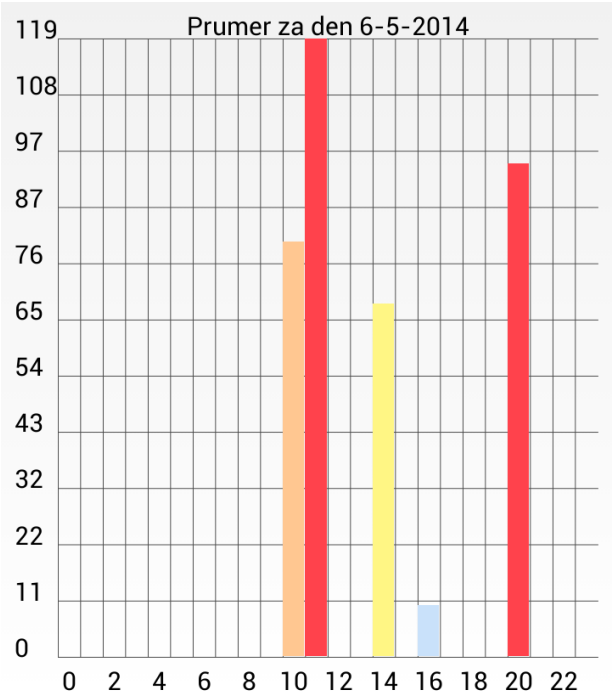
Obr. 26 Statistika testovacího dne 3.5.2014.



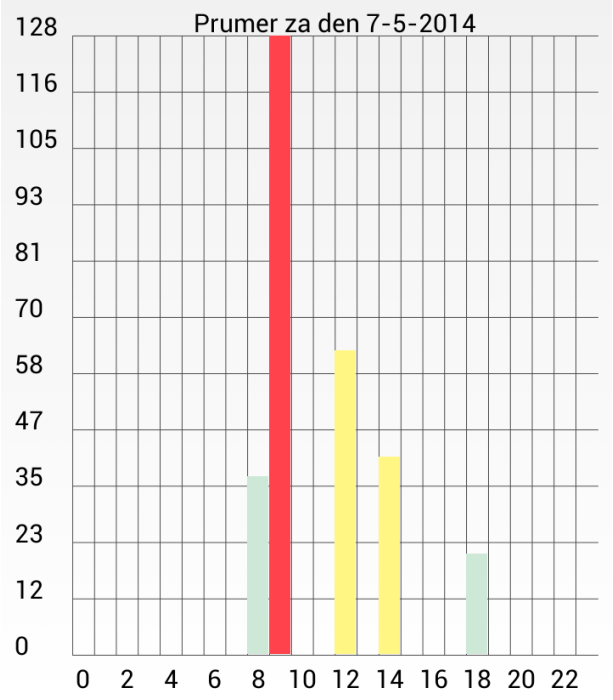
Obr. 27 Statistika testovacího dne 4.5.2014.



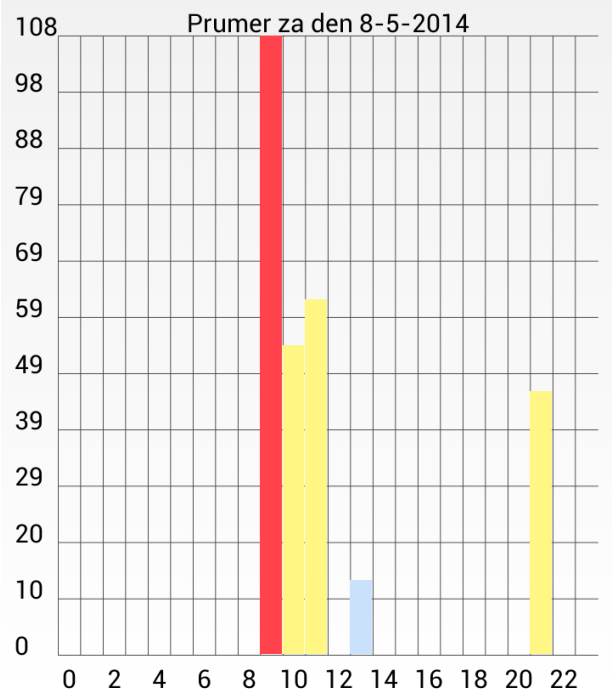
Obr. 28 Statistika testovacího dne 5.5.2014.



Obr. 29 Statistika testovacího dne 6.5.2014.



Obr. 30 Statistika testovacího dne 7.5.2014.



Obr. 31 Statistika testovacího dne 8.5.2014.

5 Závěr

5.1 Zhodnocení testování

Mobilní aplikace se při testování chovala dle očekávání, nedošlo k vyvolání žádných výjimek a správně reprezentovala veškerá data. Hardwarový prototyp *Kanárka* se také choval dle očekávání a plnil naplno svou funkci.

Hodnoty, naměřené prachovým senzorem SHARP GP2Y1010AU0F nebyly aplikací přepočítávány převodní funkcí (viz kapitola 3.4.3), protože výsledná hodnota byla nereálně malá, ve většině případů záporná. Místo toho byla data ze senzoru zpracována tak, jak byla přijata. Fakt, že výstupní napětí prachového senzoru bylo tak nízké lze vysvětlit polohou prachového senzoru v prototypu, kde jeden z jeho průduchů byl zcela zablokován nepájivým polem (viz Obr. 22), cirkulace vzduchu tedy pravděpodobně nebyla dostatečná pro měření přesných hodnot. Hodnoty z dalších *Kanárků*, stažené ze serveru Xively, byla touto funkcí před vizualizací přepočítána.

Napájení *Kanárka* 9 V alkalickou baterií se ukázalo jako velmi nevhodné a testování jasně ukázalo, že je třeba použít jiný napájecí obvod, například Li-ion nebo Li-pol bateriový systém (viz kapitola 2.1.2). Již po krátké době začalo napětí dodávané baterií klesat a posléze začalo klesat i výstupní napětí napěťových regulátorů. Jako hraniční napětí, při kterém obvod ještě dokázal fungovat, se ukázalo být cca 7 V. Pokud 9 V baterie přestala dodávat alespoň 7 V, integrované obvody přestaly fungovat.

5.2 Další práce a vývoj

Další práce na projektu musí zahrnovat dokončení mobilní aplikace. V mobilní aplikaci chybí dokončit poslední Fragment (Profil), který bude vizualizovat jednotlivá měření spolu s GPS polohou a vyobrazením na mapě. Dále chybí implementace odesílání naměřených dat na server Xively. Toho bude dosaženo přidáním dalšího podtypu *AsyncTask* ve třídě Communicator, který bude mít

odesílání na starosti. Jako třetí věc chybí implementovat přihlašovací proceduru na server www.kanarci.cz.

Mobilní aplikace by dále měla projít uživatelským testováním za účelem odhalení chyb a nedostatků v uživatelském rozhraní.

Dále je nutné optimalizovat spotřebu hardware. Je potřeba navrhnout jiný napájecí obvod (například Li-ion nebo Li-pol). Je také zapotřebí optimalizovat firmware. V současné verzi provádí *Kanárek* měření každou vteřinu a každých patnáct vteřin odesílá data a to i v případě, že aplikace neposlouchá. Vhodná by tedy byla implementace obousměrné komunikace mezi aplikací a *Kanárkem*. Aplikace by oznámila *Kanárkovi*, že začíná měřit a *Kanárek* by v tu chvíli začal napájet infračervenou LED diodu prachového senzoru, číst z něj data a odesílat je.

Dalším krokem ve vývoji projektu by měla být implementace komunikace prostřednictvím softwarového modemu. Ta by nabízela alternativu ke komunikaci prostřednictvím Bluetooth, která by sice vyžadovala dodatečný hardware, ale také nabízela podstatně menší spotřebu energie.

Dalším krokem by měl být návrh obvodu s použitím Bluetooth 4 modulu namísto použitého Bluetooth 2 modulu. Bluetooth 4 je energeticky mnohem výhodnější. Tento krok zahrnuje výměnu modulu v obvodu *Kanárka* a změnu v aplikaci, konkrétně ve třídě BluetoothService. Platforma Android poskytuje API pro Bluetooh 4 (BLE – Bluetooth Low Energy), to je ale dostupné až od verze Androidu 4.3 (API verze 18).

5.3 Celkové zhodnocení projektu

Sledování kvality ovzduší a hladiny znečištění polétavými částicemi je vzhledem ke zdravotním rizikům a dnešní produkci znečištění velice důležité a často podceňované. Projekt *Kanárci* se snaží dosáhnout dvou hlavních cílů. Prvním a možná nejdůležitějším z nich je rozšířit veřejné povědomí o důsledcích polétavého znečištění, o jeho dopadech na lidské zdraví i životní prostředí a o možnostech jeho monitorování. Druhým cílem je poskytnutí široké veřejnosti prostředek k monitorování polétavého znečištění, který by byl spolehlivý, finančně dostupný a snadno

použitelný.

V rámci tohoto projektu byl navržen a sestrojen prototyp obvodu mobilního *Kanárka*, který je bateriově napájený a výsledky měření odesílá prostřednictvím Bluetooth 2 modulu. Dále byla navržena a naprogramována mobilní aplikace pro platformu Android, která s *Kanárkem* komunikuje prostřednictvím Bluetooth, vizualizuje výsledky měření, zobrazuje denní a týdenní statistiky měření a zobrazuje mapu s vyznačenými stanicemi ČHMÚ a dalšími aktivními *Kanárky*. Cena použitých součástek nepřekročila cenovou hranici 1500 Kč a s rezervou tedy splnila cenový limit 100 Euro. Tato rezerva mimo jiné také poskytuje finanční prostředky například pro adekvátnější napájecí obvod.

Efektivita projektu mobilních *Kanárků* je přímo úměrná počtu uživatelů. Čím více aktivních zařízení hlásí výsledky, tím hustší je mapa měření znečištění. Větší potenciál však osobně vidím v rozšíření stacionárních *Kanárků*. U stacionárního zařízení odpadá potřeba optimalizace spotřeby, protože může být napájeno ze sítě. Dále také odpadá potřeba minimalizace a proto může být stacionární *Kanárek* eventuálně vybaven i dalšími senzory pro měření jiných znečišťujících látek. Takové zařízení lze pak snadno umístit například za okno, nebo i v rámci městské infrastruktury (například do parkovacích automatů). Stacionární *Kanárci* tak mohou plnohodnotně doplnit systém měřících stanic ČHMÚ.

6 Literatura

[1] United States Environmental Protection Agency: Particulate matter. [online]. Dostupné z: http://www.epa.gov/pm/

[2] Particulates. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné z: http://en.wikipedia.org/wiki/Particulates

[3] Air Info Now: Particulate matter. [online]. Dostupné z: http://www.airinfonow.org/html/ed_particulate.html

[5] European Environment Agency: Sources of air pollution. [online]. Dostupné z: http://www.eea.europa.eu/publications/2599XXX/page010.html

[6] European emission standards. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné z: http://en.wikipedia.org/wiki/European_emission_standards

[7] European Commission: Environment - Air. [online]. Dostupné z: http://ec.europa.eu/environment/air/

[8] European Environmen Agency: Spatial assessment of PM10 and ozone concentrations in Europe (2005). [online]. Dostupné z: http://www.eea.europa.eu/publications/spatial-assessment-of-pm10-and-ozone-concentrations-in-europe-2005-1

[9] Kanarci.cz. [online]. Dostupné z: http://www.kanarci.cz/news-posts/

[10] Microcontroller. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné z: http://en.wikipedia.org/wiki/Microcontroller

[11] GM Electronic: Elektrické součástky, komponenty. [online]. Dostupné z: http://www.gme.cz/lm2574n-5-0v-p330-135

[12] Softmodem. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné z: http://en.wikipedia.org/wiki/Softmodem

[13] Android Developer: Dashboards. [online]. Dostupné z: http://developer.android.com/about/dashboards/index.html?utm_source=ausdroid.net

[14] Mobile operating system. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné z: http://en.wikipedia.org/wiki/Mobile_operating_system

[15] Open Handset Alliance: Android. [online]. Dostupné z: http://www.openhandsetalliance.com/android_overview.html

[16] Nine volt battery. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné z: http://en.wikipedia.org/wiki/Nine-volt_battery

[17] Bluetooth. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné z: http://en.wikipedia.org/wiki/Bluetooth

[18] Digi-Key Corporation: Dust sensors. [online]. Dostupné z: http://www.digikey.com/product-detail/en/GP2Y1010AU0F/425-2068-ND/720164

[19] Sparkfun: Products - Bluetooth. [online]. Dostupné z: http://www.sparkfun.com/products/12577

[20] RoboCraft. [online]. Dostupné z: http://robocraft.ru/blog/electronics/1040.html

[21] James Reuben Knowles: 5V to 3.3V Logic Level Shifting Stragety Notes. [online]. Dostupné z: http://jamesreubenknowles.com/level-shifting-stragety-experments-1741
[22] Hayes command set. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné z: http://en.wikipedia.org/wiki/Hayes_command_set
[23] ElecFreaks: Freaduino UNO. [online]. Dostupné z: http://www.elecfreaks.com/wiki/index.php?title=Freaduino_UNO
[24] Sparkfun: Tutorials - Using the BlueSMiRF. [online]. Dostupné z: http://learn.sparkfun.com/tutorials/using-the-bluesmirf/hardware-hookup
[25] Wiring. [online]. Dostupné z: http://wiring.org.co/
[26] Wiring (development platform). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné z: http://en.wikipedia.org/wiki/Wiring_(development_platform)
[27] Processing (programming language). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné z: http://en.wikipedia.org/wiki/Processing_(programming_language)
[28] AirCasting. [online]. Dostupné z: http://aircasting.org/
[29] Birdi. [online]. Dostupné z: http://getbirdi.com/
[30] AirPi. [online]. Dostupné z: http://airpi.es/index.php
[31] Air Quality Egg. [online]. Dostupné z: http://airqualityegg.com/
[32] Texas Instruments: MSP430 LaunchPad Value Line Development kit. [online]. Dostupné z: http://www.ti.com/tool/msp-exp430g2
[33] JSON. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné z: http://en.wikipedia.org/wiki/JSON
[34] Attribute–value pair. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné z: http://en.wikipedia.org/wiki/Attribute%E2%80%93value_pair
[35] Android Developer: Installing the Eclipse plugin. [online]. Dostupné z: http://developer.android.com/sdk/installing/installing-adt.html
[36] Android Developer: Set Up Google Play Services SDK. [online]. Dostupné z: http://developer.android.com/google/play-services/setup.html
[37] Google Developers: Google Maps Android API v2. [online]. Dostupné z: http://developers.google.com/maps/documentation/android/start
[38] Android Developer: App structure. [online]. Dostupné z: http://developer.android.com/design/patterns/app-structure.html
[39] Android Developer: Action Bar. [online]. Dostupné z: http://developer.android.com/guide/topics/ui/actionbar.html
[40] Android Developer: Fragments. [online]. Dostupné z: http://developer.android.com/guide/components/fragments.html
[41] GraphView. [online]. Dostupné z: http://android-graphview.org/
[42] SQLite. [online]. Dostupné z: http://www.sqlite.org/about.html
[43] W3C: HTTP Status codes. [online]. Dostupné z: http://www.w3.org/Protocols/HTTP/HTRESP.html

[44] Android Developer: BluetoothDevice. [online]. Dostupné z: http://developer.android.com/reference/android/bluetooth/BluetoothDevice.html
[45] Thread (computing). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné z: http://en.wikipedia.org/wiki/Thread_(computing)
[46] Android Developer: Thread. [online]. Dostupné z: http://developer.android.com/reference/java/lang/Thread.html