ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačů

Bakalářská práce

Vývoj softwaru pro CdZnTe spektrometr pro dálková měření radiace

Filip Rychnovský
Softwarové inženýrství a technologie

Květen 2018
Vedoucí práce: Mgr. Rastislav Hodák, Ph.D.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Rychnovský** Jméno: **Filip** Osobní číslo: **435248**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Vývoj softwaru pro CdZnTe spektrometr pro dálková měření radiace

Název bakalářské práce anglicky:

Development of software for CdZnTe spectrometer for remote measurement of radiation

Pokyny pro vypracování:

Hlavním cílem Bc. práce je vytvoření ovládacího softwaru pro polovodičový CdZnTe detektor radiace vyvinutý v ÚTEF ČVUT, který bude vyčítat data z jednotlivých komponent a čidel (detektor radiace, GSM modem, GPS lokátor a čidla atmosférického tlaku, vlhkosti a teploty).
Software napsaný v C# běžící v integrovaném Mini PC bude jednak ukládat naměřená data lokálně a také poskytovat možnost sledování realtime naměřených dat pomocí webové aplikace. Tato webová aplikace běžící v Mini PC bude přístupná přes Wifi/mobilní sít a bude poskytovat přehled o aktuálním stavu celé jednotky s možností spuštění a zastavení měření.
Aplikaci napište odolnou vůči výpadku některého z čidel a také datově úspornou s ohledem na propustnost GPRS připojení.
Hlavním výsledkem Bc. práce bude vyzkoušeni funkčnosti zařízení v libovolném terénu s přírodním radiačním pozadím, tj. vytvoření souhrnné webové radiační mapy s jednotlivými daty.

Seznam doporučené literatury:

KNOLL, Glenn F. Radiation detection and measurement. 4th ed. Hoboken, N.J.: John Wiley, c2010. ISBN 04-701-3148-9.
TEBRÜGGE, Jan. Commissioning of the COBRA demonstrator and investigation of surface events as its main background. Dortmund, 2016. DOI 10.17877/DE290R-17197.
Yocto-Meteo, User's guide. Yoctopuce [online]. Geneva, 2017 [cit. 2017-04-25]. Dostupné z:http://www.yoctopuce.com/projects/yoctometeo/METEOMK1.usermanual-EN.pdf
U-blox 7 Receiver Description. U-blox [online]. Thalwil, 2013 [cit. 2017-04-25]. Dostupné z: https://www.u-blox.com/sites/default/files/products/documents/u-blox7-V14_ReceiverDescrProtSpec_%28GPS.G7-SW-12001%29_Public.pdf
MAŠEK, Petr. FastADC 100 MSPS Control Software. UTEF ČVUT, 2016.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Mgr. Rastislav Hodák, Ph.D., ÚTEF ČVUT v Praze

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **12.05.2017** Termín odevzdání bakalářské práce: **25.05.2018**

Platnost zadání bakalářské práce:
do konce letního semestru 2018/2019

Mgr. Rastislav Hodák, Ph.D.	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc.
podpis vedoucí(ho) práce		podpis děkana(ky)

Poděkování / Prohlášení

Chtěl bych poděkovat Mgr. Rastislavu Hodákovi, Ph.D., vedoucímu bakalářské práce, za odborné vedení a rady při vypracování této bakalářské práce. Dále bych chtěl poděkovat kolegům Michalu Holíkovi a Milanu Petříkovi za konzultace ohledně hardwaru detektoru. Také bych chtěl poděkovat rodině za podporu během studia.

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 16.5.2018

.....

Abstrakt / Abstract

Cílem této bakalářské práce je návrh a implementace obslužného softwaru pro detektor ionizujícího záření, který byl vyvinut v Ústavu technické a experimentální fyziky (ÚTEF) ČVUT v Praze, a který je založen na CdZnTe krystalu. Vyvinutý software umožňuje sledovat a ovládat stav všech komponent detektoru, spouštět, zastavovat a ukládat měření, zobrazovat energetické spektrum záření a mapu pozic detekovaného záření.

Klíčová slova: ionizující záření, detektor, software, FTDI, web, C#, PHP

The goal of this bachalor thesis is the development and implementation of a software for a detector of ionizing radiation developed at the Institute of Experimental and Applied Physics (IEAP) CTU in Prague. This detector is based on coplanar CdZnTe crystal. Developed software allows to monitor and control all detector components, to start, stop and save the measurement, to view energy spectrum and map of positions of detected radiation.

Keywords: ionizing radiation, detector, software, FTDI, web, C#, PHP

Title translation: Development of software for CdZnTe spectrometer for remote measurement of radiation

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

Tato bakalářská práce se zabývá návrhem a implementací softwaru pro detektor ionizujícího záření, který byl vyvinut v Ústavu technické a experimentální fyziky (ÚTEF) ČVUT v Praze, a který je založen na CdZnTe krystalu. V kapitole Koplanární CdZnTe detektor radiace je popsán jednak princip funkce CdZnTe krystalu jako detektoru ionizujícího záření a také hardwarová struktura detektorové jednotky, s důrazem na popis komunikačních protokolů jednotlivých komponent. Kapitola Návrh softwaru je věnována návrhu softwarové architektury. Software je rozdělen na dva velké celky, část zajišťující vlastní ovládání detektoru je popsána v kapitole Implementace softwaru v detektoru a část zajišťující vzdálený přístup přes mobilní sít je popsána v kapitole Implementace softwaru na serveru. Kapitola Testování softwaru popisuje kalibraci detektoru a ukazuje funkčnost softwaru v laboratorním i reálném (terénním) měření.

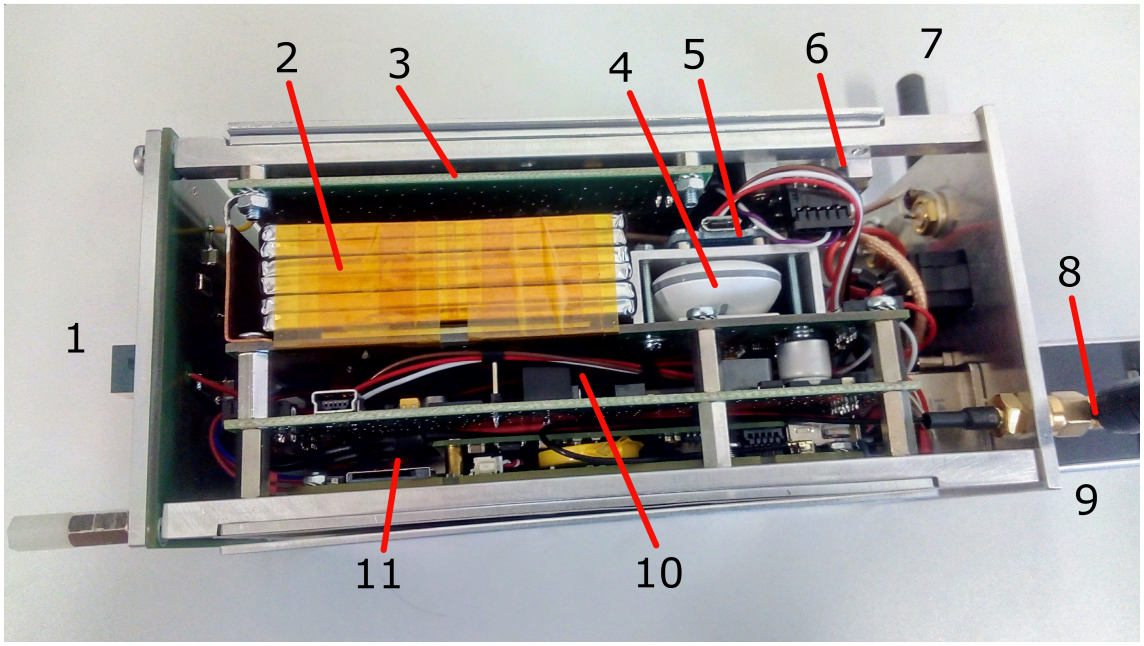
Kapitola 2
Koplanární CdZnTe detektor radiace

2.1 Princip detekce ionizujícího záření

Koplanární CdZnTe krystal je polovodičový detektor, ve kterém při interakci ionizujícího záření s citlivým objemem detektoru, vzniknou volné nositele náboje (pár elektron - díra). Aplikací elektrického pole (BIAS napětí) jsou elektrony sebrány na kladné elektrodě (anodě), čímž vznikne měřitelný elektrický impuls zpracovávaný elektronikou detektoru [1–2].

2.2 Popis hardwaru detektoru

Hardwarová část CdZnTe detektoru byla navrhnuta a vyvinuta kolegy z ÚTEF ČVUT v Praze. Jedná se o přenosný a kompaktní přístroj, který obsahuje CdZnTe krystal s jednoduchým spektrometrem, desku zdrojů s mikrokontrolerem, baterii a minipočítač ZOTAC. Vnitřní uspořádání detektoru je na obrázku 2.1. Jednotlivé komponenty jsou uloženy v hliníkovém obalu (obrázek 2.2), na jehož přední konektorové straně (obrázek 2.3) jsou umístěny: hlavní vypínač, antény Wifi a GSM modulu, GPS modul a napájecí a nabíjecí konektory.



Obrázek 2.1. Detektor shora odkrytovaný: 1 - CdZnTe krystal, 2 - baterie, 3 - analogová elektronika krystalu, 4 - GSM modul, 5 - Meteo modul, 6 - převodník FastADC, 7 - anténa Wifi, 8 - anténa GSM, 9 - GPS modul, 10 - deska zdrojů, 11 - minipočítač ZOTAC

Obrázek 2.2. Vnější pohled na detektor

Obrázek 2.3. Přední panel detektoru

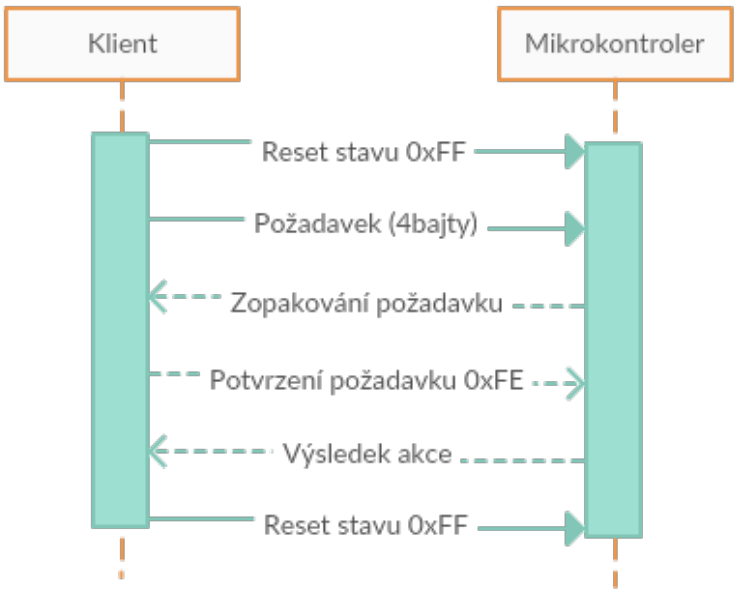
2.2.1 Mikrokontroler desky zdrojů

Mikrokontroler je umístěn na desce zdrojů, která má na starosti napájení celého přístroje, přepínání mezi baterií a externím napájením. Deska poskytuje několik napětí podle požadavků komponent: 5V (napájení USB sběrnic a počítače), 12V (napájení zesilovače), 200V napájení detektoru, 0-2000V řiditelné vysoké napětí pro detektor. Mikrokontroler je připojený přes sériovou linku a FTDI čip do USB portu počítače. Přes tento sériový port zpřístupňuje informace o stavu baterie a zdroje vysokého napětí a umožňuje řídit výstupní napětí zdroje vysokého napětí pomocí jednoduchého protokolu.

Komunikaci otevře klient zasláním resetovacího bajtu 0xFF, následné zašle požadavek o délce 4 bajtů (struktura požadavku uvedena v tabulce 2.1), mikrokontroler požadavek zopakuje a klient potvrdí správnost zasláním potvrzovacího bajtu 0xFE, mikrokontroler provede požadovanou operaci a vrátí odpověď o délce 4 bajtů s výsledkem operace. Komunikace je ukončena zasláním resetovacího bajtu 0xFF. Sekvenční diagram požadavku je na obrázku 2.4.

	Bajty požadavku				Bajty odpovědi			
Příkaz	1	2	3	4	1	2	3	4
čtení hodnoty	3	id registru	0	0	status (0-1)	MSB	LSB	0
zápis hodnoty	4	hodnota	0	0	status (0-1)	0	0	0

Tabulka 2.1. Protokol mikrokontroleru



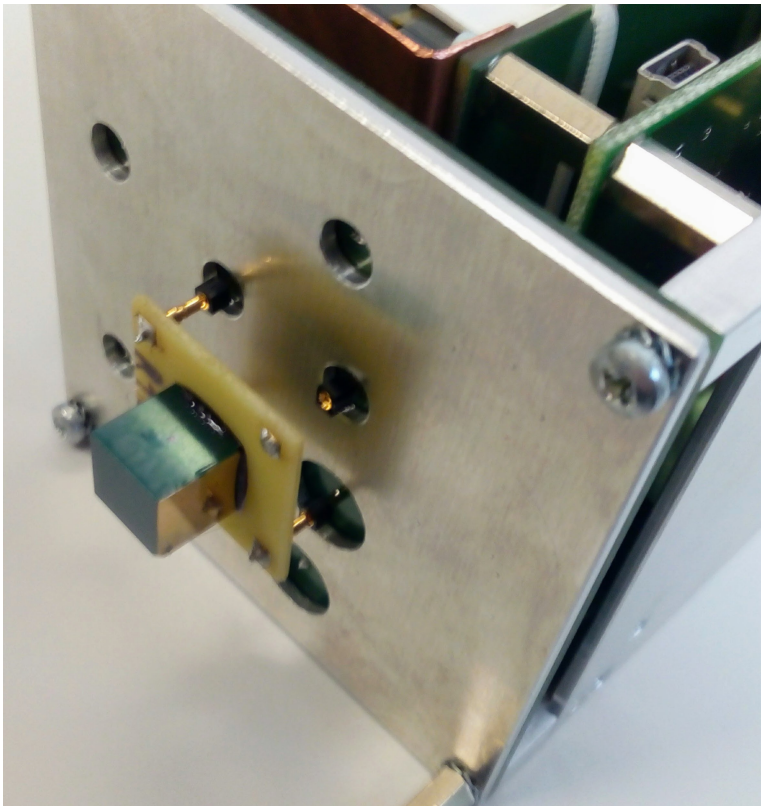
Obrázek 2.4. Sekvenční diagram požadavku na mikrokontroler

2.2.2 CdZnTe krystal a jeho elektronika

Do CdZnTe krystalu (obrázek 2.5) je připojeno napětí 200V a měnitelné napětí 0-2000V, dva výstupy jsou přes stabilizátor a zesilovač přivedeny do převodníku FastADC 100 MSPS, který analogový signál (změna napětí) převádí na digitální signál. Převodník je do počítače zapojen pomocí sériové linky a FTDI čipu do USB portu počítače.

Převodník začíná snímat signál buď při povelu od externího spouštěče nebo při nárůstu signálu nad nastavený threshold. Snímání je ukončeno po nasbírání zadaného množství vzorku nebo při poklesu signálu pod zadaný threshold. Další důležitou volbou je způsob výstupu, prvním je Waveform, kdy jsou vypsány všechny nasbírané vzorky, druhým je Sample, kdy je vypsáno maximum signálu a posledním je Sample+TOT, kdy je vypsán maximum signálu a doba, po kterou byl signál nad thresholdem.

Komunikace s převodníkem probíhá pomocí odesílání zpráv po sériové lince o dvou bajtech, kde první bajt udává číslo registru a druhý hodnotu, na kterou se má příslušný registr nastavit. Po připojení převodníku se sběr dat zakáže nastavením registru 191



Obrázek 2.5. CdZnTe krystal

na hodnotu 0 a převodník se vyresetuje pomocí nastavení registru 145 na hodnotu 0 a následně nastavíme všechny konfigurační registry uvedené v tabulce 2.2.

Sběr dat převodníkem je zahájen povolením sběru dat nastavením registru 141 na hodnotu 0 a poté vyresetováním převodníku pomocí nastavení registru 145 na hodnotu 0. Převodník poté podle konfigurace zpracovává analogový signál a naměřené hodnoty posílá v paketu, jehož struktura je uvedená v tabulce 2.3. Sběr dat se ukončí zakázáním sběru dat, tedy nastavením registru 191 na hodnotu 0. Celý proces komunikace je zachycen na sekvenčním diagramu na obrázku 2.6.

2.2.3 Meteorologický senzor

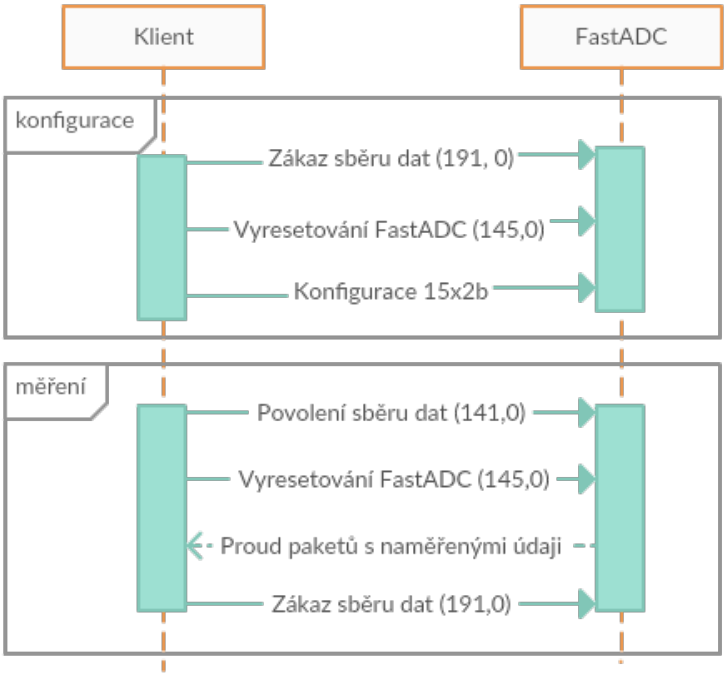
Pro sledování vnitřních podmínek je v detektoru umístěn meteorologický sensor Yocto-Meteo od firmy Yoctupuce [3], který je do počítače zapojen přes USB. Výrobce poskytuje knihovny pro komunikaci se senzorem ve všech běžných jazycích.

2.2.4 GPS senzor

Do USB portu na předním panelu je zapojený GPS modul Canmore GT-730F. Tento modul umožnuje získání GPS pozice jak pomocí protokolu NMEA 0183 přes sériový port, tak také pomocí součásti operačního systému Windows: Windows Sensor and Location platform [4].

adresa registru	význam registru
128	dolní treshold MSB (MSB=most significant byte)
129	dolní treshold LSB (LSB=least significant byte)
130	horní treshold MSB
131	horní treshold LSB
132	hystereze (kompezacace roztřepenosti signálu)
133	clock divider (frekvence vzorkování)
134	počet vzorku uložený před trigerem MSB
135	počet vzorku uložený před trigerem LSB
136	počet vzorku uložených mezi trigery MSB
137	počet vzorku uložených mezi trigery LSB
138	počet vzorku uložených po trigeru MSB
139	počet vzorku uložených po trigeru LSB
140	operační mod 1 bit 0 2-4 bit: 000 Sample, 001 SampleTOT, 100 Waveform 5-6 bit počátek pulzu: 00 překročení thresholdu, 01 externí trigger 7-8 bit konec pulzu: 00 pokles pod threshold, 01 po počtu událostí
142	konfigurace trigeru 1-3 bit 100 - go through 4 bit polarita signalu (1 = +, 0 = -) 5-8 bit 0000 externi trigger ani gate není použit
143	formát datového paketu 1 bit přidání hlavičky 2 bit přidání časové značky 3 bit přidáná controlního součtu 4-7 bit 0000 8 bit 1 použita 8bit sběrnice

Tabulka 2.2. Konfigurace FastADC



Obrázek 2.6. Sekvenční diagram protokolu převodníku FastADC

byte	popis
1	0xFF
2	0xF0
3	1-3bit 000 1bit modout 5-8bit index
4	0x0F
5	počet datových položek MSB
6	počet datových položek LSB
7+2*i	datová položka MSB
7+2*i+1	datová položka LSB
7+2*length	timestamp 1byte
7+2*length +1	timestamp 2byte
7+2*length +2	timestamp 3byte
7+2*length +3	timestamp 4byte
7+2*length +4	timestamp 5byte
7+2*length +5	timestamp 6byte
7+2*length +6	kontrolní součet MSB
7+2*length +7	kontrolní součet LSB

Tabulka 2.3. Struktura datového paketu FastADC

Kapitola 3
Návrh softwaru

3.1 Katalog požadavků

Ze zadání bakalářské práce vyplývají tyto požadavky:

3.1.1 Funkční požadavky

- Uživatel může spustit měření
- Uživatel může zastavit měření
- Uživatel může zobrazit mapu s aktuálně měřenými hodnotami
- Uživatel může zobrazit celou historii měření
- Měřením se rozumí vyčítání dat z detektoru radiace, GPS přijímače a čidla atmosférického tlaku, vlhkosti a teploty
- Při provozu na baterii musí aplikace při nízkém stavu baterie včas ukončit práci

3.1.2 Nefunkční požadavky

- Aplikace poběží ve Windows Mini PC
- Uživatelské rozhraní bude řešeno webovou stránkou
- Webová stránka musí být přístupná při připojení na integrovanou Wifi
- Webová stránka musí být přístupná přes mobilní sít
- Aplikace musí být odolná proti výpadku komponent
- Webová stránka musí být datově úsporná (připojení GPRS)

3.2 Architektura softwaru

Po analýze požadavků bylo zvoleno rozdělení na tři velké samostatné celky: webová aplikace na serveru, webové rozhraní na detektoru a měřící služba na detektoru. Webové rozhraní detektoru s měřící službou budou spolu komunikovat pomocí pojmenovaných rour v rámci technologie Windows Communication Foundation (WCF), která umožňuje volání metod publikovaných druhou aplikací v jejím manifestu. Jako hlavní úložiště dat na detektoru byla zvolena SQLite databáze hlavně pro svoji výkonovou nenáročnost, na serveru je použita MySQL.

3.2.1 Webová aplikace na detektoru

Webové rozhraní v podobě aplikace psané v C# a ASP.NET běžící v detektoru bude zobrazovat aktuální stav detektoru, tedy stav všech komponent, spektrum aktuálního měření, možnost nastavení BIAS napětí, spuštění měření v kalibračním režimu, bez GPS nebo s GPS, zastavení měření s i bez uložení, zobrazení uložených měření, na základě kalibračního měření možnost vypočítat kalibrační konstanty A a B. Historie měření bude načítána ze SQLite databáze pomocí Entity Frameworku. Na vykreslování spekter měření bude použita knihovna JSROOT. Ostatní operace jsou předávány druhé části aplikace – tedy měřící službě pomocí výše zmíněných pojmenovaných rour.

3.2.2 Měřící služba

Měřící služba je koncipována jako Windows služba, která v sobě hostí samostatné služby technologie WCF pro každou komponentu.

FastADC služba má na starosti komunikaci s převodníkem FastADC přes sériovou linku, tedy otevření spojení, konfigurace převodníku, spuštění a zastavení sběru dat a vyčítání datových paketů.

PowerBoard služba má na starosti komunikaci s mikrokontrolerem na desce zdrojů přes sériovou linku, tedy nastavení a vyčítání aktuálního a nastaveného BIAS napětí a napětí baterie, z čehož určuje úroveň jejího nabití.

Meteo služba má na starosti komunikaci s meteorologickým senzorem přes knihovnu výrobce a vyčítání tlaku, teploty a vlhkosti.

GPS služba má na starosti zjišťování aktuální polohy pomocí systémové služby Sensor and Location platform.

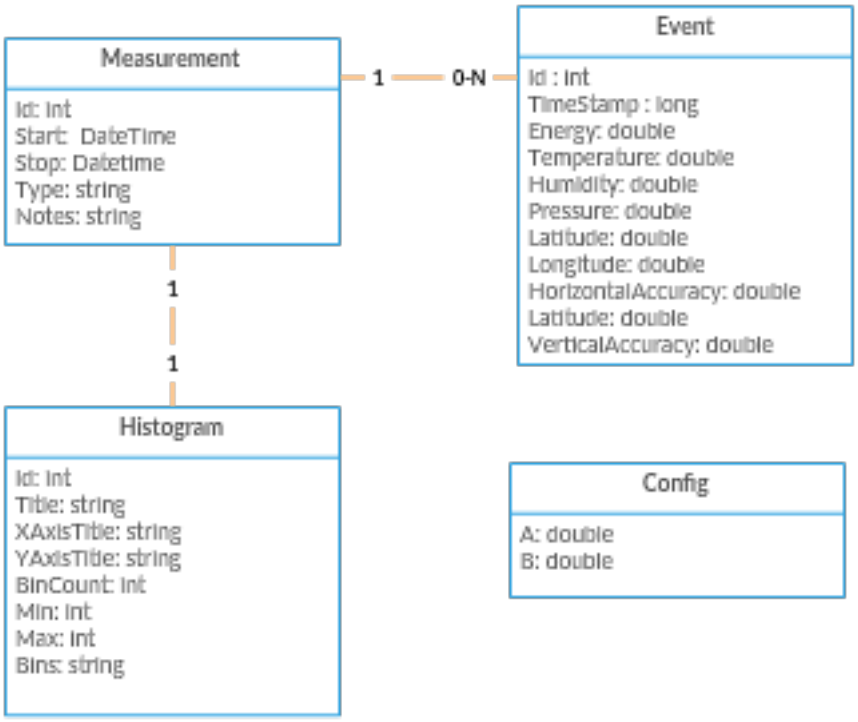
Měřící služba má na starosti start měření, vyčítání dat z FastADC, Meteo a GPS služby, tvorbu logu událostí a energetického spektra záření. Z důvodu rychlosti bude mít několik vláken, jedno bude mít na starosti pravidelné vyčítání meteorologických dat z Meteo služby, další pravidelné vyčítání GPS pozice z GPS služby, další vyčítání nového detekovaného záření z FastADC služby a přidání zkalibrovaných energií do fronty, ze které poslední vlákno spojuje detekované záření s časovým razítkem, meteorologickými daty, a případně GPS pozice a přidá ji do logu a energetického spektra záření. Při ukončení měření, podle volby uživatele, naměřená data uloží pomocí Entity Frameworku do SQLite databáze.

3.2.3 Server

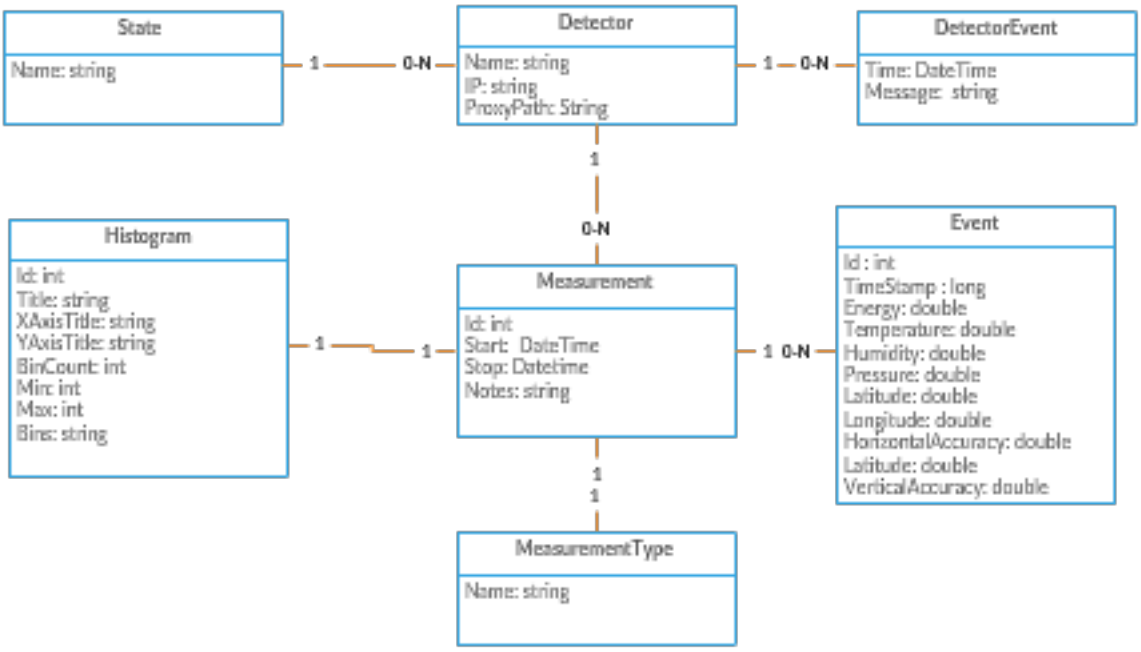
Virtuální server běžící v clusteru ÚTEF bude plnit dvě úlohy, jednak pomocí VPN a reverzní proxy bude vzdáleně zpřístupňovat webová rozhraní detektorů, za druhé jeho webové rozhraní psané v PHP a Nette frameworku s MVC architekturou bude zobrazovat stav jednotlivých detektorů a umožní prohlížení dat z měření provedených jednotlivými detektory a také zobrazení souhrnné mapy naměřeného záření ze všech uskutečněných měření. K přístupu k MySQL databázi bude použito knihovny Doctrine.

3.3 Datový model

Na základě požadavků byly navrženy dva datové modely, jeden pro aplikaci na detektoru (obrázek 3.1), který bude ukládán v lokální databázi SQLite na detektoru, a druhý pro serverovou část aplikace (obrázek 3.2) který bude ukládán v databázi MySQL.



Obrázek 3.1. Datový model detektoru



Obrázek 3.2. Datový model serveru (z důvodu úspory místa zde nejsou uvedeny podtřídy tříd MeasurementType (Calibration, Inside, Outside), DetectorState (Ready, Offline, Error, Measuring) a DetectorEvent (BreakDown, Fixed, Connected, Disconnected, MeasurementStarted, MeasurementEnded)

Kapitola 4
Implementace softwaru detektoru

Na základě architektury popsané v předchozí kapitole byl naimplementován software pro detektor o třech celcích: webová aplikace, měřící služba a knihovna pro třídy použité v obou předchozích částech. Implementace softwaru na serveru pro vzdálený přístup bude popsána v další kapitole.

4.1 Společná knihovna

Je zde jmenný prostor `Common.ServiceContracts`, který obsahuje definici všech pěti rozhraních měřící služby přístupné přes pojmenovanou linku. Dále je zde jmenný prostor `Common.Model`, který obsahuje databázové entity.

Třída `Common.MeasurementContext` zajištuje všechny databázové operace (čtení, vkládání, úpravu a mazaní entit) přes Entity Framework.

Třída `Common.WCFClients` obsahuje továrničky na klienty všech pěti pojmenovaných rour zpřístupněné měřící službou. Obsahuje metody `GetFastADCServiceClient()`, `GetMeasurementServiceClient()`, `GetPowerBoardServiceClient()`, `GetMeteoServiceClient()` a `GetGPSServiceClient()`. Vzhledem k tomu, že klienty vytváří WCF na základě servisních kontraktů, které jsou popsány v další sekci a URI, na které služba poslouchá, tak tyto metody volají generickou továrníčku `GetClient<T>(string namedPipe)`.

4.1.1 ServiceContracts

Interface `IFastADCService` definuje vnější rozhraní FastADC služby, tedy `GetStatus()` vracející aktuální stav této služby, `Start()` spouští sběr dat, `Stop()` zastaví sběr dat a `GetEnergy()` vrací energii další částice.

Interface `IPowerBoardService` definuje vnější rozhraní PowerBoard služby, tedy `GetStatus()` vracející aktuální stav této služby, `GetSetVoltage()` vracející nastavené BIAS napětí, `GetCurrentVoltage()` vracející aktualní BIAS napětí, `GetBateryLevel()` vracející procenta nabití baterky, `GetState()` vracející všechny předchozí údaje v jednom WCF požadavku a `SetVoltage(int)` nastavující BIAS napětí .

Interface `IGPSService` definuje vnější rozhraní GPS služby, tedy `GetStatus()` vracející aktuální stav této služby, `GetPosition()` vrací aktuální GPS pozici, `GetState()` vracející polohu a stav v jednom WCF požadavku, `Start()` spouští získávání polohy a `Stop()` zastavuje sběr polohy.

Interface `IMeteoService` definuje vnější rozhraní Meteo služby, tedy `GetStatus()` vracející aktuální stav této služby, `GetMeteoData()`, která vrací aktuální meteorologické údaje a `GetState()` vracející meteorologické údaje a stav v jednom WCF požadavku.

Interface `IMeasurementService` definuje vnější rozhraní Measurement služby, tedy `GetStatus()` vracející aktuální stav této služby, `GetMeasurementType()` vracející typ aktuálního měření, `StartRawMeasurement()` zahajující měření bez použití kalibrace, `StartMeasurement(double, double)` zahajující měření se zadanou kalibrací,

`StartMeasurementWithGPS(double, double)` zahajující měření se zadanou kalibrací a GPS, `StopWithoutSave()` ukončující měření bez uložení, `StopWithSave(string)` ukončující a ukládající měření s volitelnou poznámkou, `GetHistogram()` - vracející histogram aktuálního měření, `GetMobileHistogram()` - vracející datově redukovaný histogram aktuálního měření a `GetDataLog()` vracející log aktuálního měření.

4.1.2 Model

Tento jmenný prostor obsahuje všechny databázové entity s jejich atributy, tedy třídy `Config`, `Measurement`, `Event` a `Histogram` odpovídající navrženému datovému modelu na obrázku 3.1 v předchozí kapitole.

4.2 Měřící služba

Vstupním bodem této části aplikace, je třída `WindowsService`, jejímž úkolem je nastartování 5 služeb technologie WCF tedy `FastADCService`, `PowerBoardService`, `GPSService`, `MeteoService` a `MeasurementService` v jedné Windows službě. Vzhledem ke komplikovanosti registrace Windows služeb, je pro účely ladění implementováno v třídě `WindowsService` spuštění bez nutnosti registrace Windows služby. V následujících sekcích budou popsány jednotlivé služby.

4.2.1 FastADCService

Třída `FastADCService` implementuje výše zmíněný servisní kontrakt `IFastADCService` a odpovídá na WCF požadavky. V konstruktoru služby předá pokyn třídě `FastADC`, aby se připojila k FastADC převodníku a provedla jeho konfiguraci. Metoda `GetStatus()` vrací hodnotu členské proměnné status, která vyjadřuje aktuální stav služby i převodníku. V metodě `Start()` je třídou `FastADC` zahájeno měření a v metodě `Stop()` je třídou `FastADC` měření ukončeno. Metoda `GetEnergy()` vrací energii dalšího záření. V destruktoru je provedeno odpojení od FastADC převodníku.

Třída `FastADC` je zodpovědná za high level operace s FastADC. V konstruktoru je přijata konfigurace v podobě objektu třídy `FastADC.Config`. Pro operace na sériové lince používá třídu `FastADC.Driver`. Metoda `ConnectAndConfigure()` má na starost připojení a konfiguraci podle protokolu zmíněného v kapitole 2, tedy otevření sériové linky ovladačem, zaslání příkazu zákazu sběru a reset zařízení a odeslání konfigurační zprávy dle třídy `FastADC.Config`. `StartMeasurement()` vyresetuje převodník a povolením sběru dat zahájí měření. `StopMeasurement()` zakázáním sběru dat ukončí měření. Metoda `Next()` přečte další paket a vrátí energii záření přičemž posune rozsah, z kdy -V je 0, 0V je 32687 +V je 65535 na -V je -32688, 0V je 0, a +V je 32687. `SendResetCommand()` zašle převodníku příkaz na vyresetování. `SendSetInhibitCommand()` zašle převodníku příkaz na zakázání sběru dat. `SendClearInhibitCommand()` zašle převodníku příkaz na povolení sběru dat.

Třída `FastADC.Driver` je zodpovědná za low level operace s převodníkem FastADC. Pro komunikaci s převodníkem přes sériovou linku připojenou do počítače FTDI převodníkem, užívá FTDI knihovnu. V metodě `Connect(string desc)` se provede otevření spojení přes FTDI čip podle zadaného jména. Nastaví se parametry sériové linky tzn. baud rate 2000000, 8 datových bitu, jeden stop bit, bez parity. V metodě `Configure(Config config)` se provede odeslání konfigurace (kterou tvoří 15 dvojic, adresa registru a jeho hodnota) do převodníku. `GetNextDataPacket()` vrací další datový paket, nejprve počká, až je na lince alespoň celá hlavička tzn.

6 bajtů, použije třídu `Packet` k validaci hlavičky a výpočtu, kolik má celý paket bajtů. Přečte potřebný kus dat a předá ho třídě `Packet` k parsování. Výsledný přečtený a zparsovaný paket vrátí. `GetBufferSize()` vrátí počet dat v přijímacím bufferu FTDI čipu. `WriteBytes(byte[] data)` odešle pole bajtů po sériové lince. `ReadBytes(uint length)` přečte daný počet bajtů ze sériové linky. `ExecuteComnand(Request request)` vykoná zadaný příkaz, tedy odešle dvojici bajtu tvořenou adresou registru a jeho hodnotou.

`FastADC.Packet` je třída reprezentující datový paket FastADC, je odpovědná za jeho parsování a validaci. Metoda `ParseHeader(byte[] header)` ověří validnost formátu 6 bajtové hlavičky a přečte z ní index paketu a délku datové části paketu. Z délky datové části spočítá délku těla paketu. `ParseBody(byte[] body)` zkontroluje, zda sedí kontrolní součet (poslední 2 bajty paketu) a parsuje 2 bajtové hodnoty v datové části a 6 bajtový timestamp.

Třída `FastADC.Request` reprezentuje 2 bajtový příkaz (adresa registru a jeho hodnota), obsahuje enum s adresami všech používaných registrů.

Úkolem třídy `FastADC.Config` je na základě hodnot v konfiguračním souboru app.config sestavit konfigurační paket. Konfigurační soubor obsahuje tyto položky:

- **polarita** – zde se uvádí polarita vstupního signálu, hodnoty positive, negative
- **operationMode** – zde se uvádí operační mód převodníku, hodnoty sample_threshold, sample_trigger, sample_tot_threshold, sample_tot_trigger, waveform_trigger_count, waveform_trigger_threshold, waveform_threshold_threshold, waveform_threshold_count
- **hystereze** – zde se uvádí, jaká roztřepenost signálu se má pro účely překročení thresholdu ignorovat
- **clock devider** – použito jen při módu Waveform – každý N vzorek bude zaznamenán
- **pretrig** – použito jen při módu Waveform – udává, kolik vzorků před počátkem pulzu má být uloženo
- **count** – použito jen při módu Waveform při ukončovací události count - udává, kolik vzorků pulzu má být zaznamenáno
- **posttrig** – použito jen při módu Waveform - udává, kolik vzorků po konci pulzu má být uloženo
- **low_threshold** – dolní threshold, při jejím překročení je pulz zaznamenán, hodnoty 0-65535, 0V je 32687
- **up_threshold** – horní threshold, při jejím překročení není pulz zaznamenán, hodnoty 0-65535, 0V je 32687

Tyto hodnoty jsou převedeny do 15 dvojic podle tabulky 2.2.

4.2.2 PowerBoardService

Třída `PowerBoardService` implementuje výše zmíněný servisní kontrakt `IPowerBoardService` a odpopovídá na WCF požadavky. K provedení samotných operací na mikrokontroleru používá třídu `Controler`. V konstruktoru se zavoláním `Controler.Connect()` otevře spojení s mikrokontrolerem. Metoda `GetStatus()` vrací hodnotu členské proměnné status, která vyjadřuje aktuální stav služby i mikrokontroleru. Metoda `GetCurrentVoltage()` vyčte metodou `Controler.GetHWValue()` aktuální napětí zdroje vysokého napětí, nebo pokud právě probíhá postupné nastavení napětí, tak vrátí poslední známou hodnotu. Metoda `GetSetVoltage()` vyčte metodou `Controler.GetHWValue()` nastavené napětí zdroje vysokého napětí. Metoda `GetBaterryLevel()` vyčte procenta kapacity baterie pomocí metody `Controler.GetBateryPercentage()`. Metoda `GetState()` vrací hodnoty předchozích

metod v jednom WCF požadavku. Metoda `SetVoltage(int voltage)` vydá třídě `Controler` pokyn k postupnému zvýšení/snížení napětí zdroje vysokého napětí na požadovanou hodnotu zavoláním `Controler.SetHWSafely(voltage)`. Při ukončování služby je provedeno odpojení v metodě `Dispose()`.

Třída `PowerBoardControler.Controler` je odpovědná za high level operace s mikrokontrolerem. Pro operace na sérové lince používá třídu `PowerBoardControler.Driver`. Metoda `Connect()` a `Disconnect()` volá stejnojmennou metodu ve třídě `Driver`. Metoda `GetHWValue()` získá aktuální hodnotu BIAS napětí. Pro přepočet hodnoty registru na hodnotu vysokého napětí se používá přepočet, kde ReferenceVolts je napětí mikrontroleru (4.096 V) a AdcValue je hodnota registru:

$$CurrentHighVoltage = \frac{ReferenceVolts * AdcValue * 1024}{1000}$$

Metoda `GetSetHWValue()` získá aktuální nastavenou hodnotu BIAS napětí. Pro přepočet hodnoty registru na hodnotu vysokého napětí se používá přepočet, kde HighVoltageMaximum je maximální dovolené napětí zdroje vysokého napětí (2000 V) a MaximumVolts je maximální napětí mikrokontroleru (4.096V) :

$$SetHighVoltage = \frac{HighVoltageMaximum * ReferenceVolts * AdcValue * 1024}{MaximumVolts * 1000}$$

Metoda `GetBatteryVoltage()` získá aktuální napětí baterie. Pro přepočet hodnoty registru na napětí baterie se používá přepočet, kde deviderA (20.6 kΩ) a deviderB (6.8 kΩ) jsou hodnoty odporů napěťového děliče :

$$BatteryVoltage = \frac{ReferenceVolts * AdcValue * 1024 * (deviderA + deviderB)}{1000 * deviderB}$$

Metoda `GetBatteryPercentage()` přepočítává napětí baterky z metody `GetBatteryVoltage()` na odhad procenta kapacity baterie (Li-Pol 3s 11.1 V, 5000 mAh), pomocí vzorce, který byl odvozen z vybíjecí křivky baterie (12 V v zátěži odpovídá 100%, 10,4 V odpovídá 20%), která je v střední části lineární a proto byly vypočteny hodnoty A na 50 a B na -500:

$$BatteryPercentage = BatteryVoltage * A + B$$

Výše uvedené metody vyčítají hodnotu příslušného registru pomocí metody `GetADCValue(Adcs adc)`,kde Adcs je enum čísel registrů (2: stav baterky, 3: aktuální napětí zdroje vysokého napětí, 4: nastavené napětí zdroje vysokého napětí), pomocí třídy `PowerBoardControler.Request` sestaví požadavek, který vyšle pomocí `ExecuteCommand(Request request)` třídy `Driver`. Z odpovědi přečte požadovanou hodnotu a vrátí ji. Metoda `SetHWSafely(int voltage)` nastavuje napětí na požadovanou hodnotu pomocí metody `SetHW(int voltage)` po malých 25V skocích. Pro přepočet hodnoty vysokého napětí na hodnotu registru v požadavku je použit vzorec:

$$DACValue = \frac{voltage * MaximumVolts * 256}{HighVoltageMaximum * ReferenceVolts}$$

Po každé změně čeká na ustálení napětí na požadovanou hodnotu pomocí metody `WaitHWStabilization(int targetvoltage)`, která čeká až skutečné napětí bude blízké nastavenému. Toto čekání je implementováno, protože změna napětí zdroje vysokého napětí není okamžitá (např. z 1500V na 0V trvá několik minut).

Třída `PowerBoardControler.Driver` je odpovědná za low level operace s mikrokontrolerem. Pro komunikaci s mikrokontrolerem přes sériovou linku připojenou do počítače FTDI převodníkem užívá FTDI knihovnu. V metodě `Connect(string desc)` se provede otevření spojení přes FTDI čip podle zadaného jména a nastaví se parametry sériové linky tzn. baud rate 9600, 8 datových bitu, jeden stop bit, bez parity. Metoda `ExecuteCommand(Request request)` provede požadavek podle protokolu na obrázku 2.4) v předchozí kapitole, tedy vyresetování mikrokontroleru pomocí zaslání 0xFF, zaslání 4 bajtového požadavku, ověření správnosti zopakovaného požadavku a potvrzení požadavku zasláním 0xFE a přečtení odpovědi. K zápisu a čtení dat na sériové lince používá metody `WriteBytes(byte[] data)` a `ReadBytes(uint length)`.

Třída `PowerBoardControler.Request` složí 4 bajtový požadavek z typu požadavku (3 - čtení hodnoty, 4 - zápis hodnoty) a hodnoty, což je pro čtení číslo čteného registru a pro zápis je to hodnota napětí. Pro účely validace zopakovaného požadavku mikrokontrolerem převede 4 bajtové pole na požadavek a metodou `Equals` umožňuje jejich porovnání.

Třída `PoweBoardControler.Response` z 4 bajtové odpovědi, přečte status a případnou hodnotu požadovaného registru.

4.2.3 GPSService

Třída `GPSService` implementuje výše zmíněný servisní kontrakt `IGPSService` a odpovídá na WCF požadavky. Pro komunikaci s přijímačem je použito systémové rozhraní Windows Sensor and Location Platform, které poskytuje jednotný přístup k nejrůznějším senzorům zapojených do počítače. Toto rozhraní je v C# dostupné přes systémovou knihovnu `System.Device.Location` konkrétně její třída `GeoCoordinateWatcher`. V konstruktoru služby je tato knihovna inicializována, ale vzhledem k tomu, že GPS se nepoužívá u některých typů měření, tak se start sběru polohy nezahajuje. Metoda `GetStatus()` vrací status podle stavu třídy `GeoCoordinateWatcher`. `GetPosition()` vrací aktuální pozici. Metoda `GetState()` vrací stav i polohu v jednom WCF požadavku. Metoda `Start()` zahájí získávání polohy třídou `GeoCoordinateWatcher` a metoda `Stop()` získávání polohy ukončí. Metoda `Dispose` volá metodu `Dispose` třídy `GeoCoordinateWatcher`.

4.2.4 MeteoService

Třída `MeteoService` implementuje výše zmíněný servisní kontrakt `IMeteoService` a odpopovídá na WCF požadavky. V konstruktoru se pomocí třídy `Meteo` provede připojení k meteorologickému sensoru. Metoda `GetStatus()` vrací hodnotu členské proměnné status, která vyjadřuje aktuální stav služby i sensoru. Metoda `GetMeteoData()` vrátí aktuální meteorologické údaje z třídy `Meteo`. Metoda `GetState()` vrací aktualní stav a meteorologické údaje v jednom WCF požadavku.

Třída `Meteo` zajišťuje komunikaci s meteorologickým senzorem pomocí knihovny Yoctopuce API od výrobce senzoru. V konstruktoru definujeme režim přístupu k čidlům meteorologického sensoru – v našem případě přes USB, další možností je několik síťových variant propojení. Poté získáme voláním `YPressure.FirstPressure()`, `YHumidity.FirstHumidity()` a `YTemperature.FirstTemperature()` objekty, pomocí kterých budeme komunikovat s jednotlivými čidly. V metodě `GetMeteoData()` se vrací aktuální hodnoty ze všech čidel, které získáme zavoláním `get_currentValue()` na objektech čidel.

4.2.5 MeasurementService

Třída `MeasurementService` implementuje výše zmíněný servisní kontrakt `IMeasurementService` a odpovídá na WCF požadavky. Tato služba je zodpovědná za vlastní měření, tedy spojování údajů z `FastADCService`, `MeteoService` a `GPSService`. V konstruktoru této služby si pomocí metod z třídy `Common.WCFClients` inicializuje klienty zmíněných služeb. Metoda `GeTStatus()` vrací hodnotu členské proměnné status, která vyjadřuje aktuální stav služby a měření. `GetMeasurementType()` vrací typ aktuálního měření. Metoda `StartRawMeasurement()` zahájí kalibrační měření (bez kalibrace), `StartMeasurement(double a, double b)` zahájí měření s kalibračními parametry A a B, `StartMeasurementWithGPS(double a, double b)` zahájí měření s kalibračním parametry A a B a zapnutou GPS. Měření znamená, že se spustí 5 vláken. První vlákno má na starosti každou sekundu aktualizovat meteorologické údaje z `MeteoService`. Druhé vlákno má na starosti každou sekundu aktualizovat polohu z `GPSService`. Třetí vlákno má na starosti do fronty ukládat naměřené záření z `FastADCService`. Čtvrté vlákno má na starosti čtení záření z fronty a zapisování zkalibrované hodnoty do logu událostí (s meteorologickými údaji a případnou polohou) a do histogramu. Páté vlákno každou minutu přečte stav baterie z `PowerBoardService` a pokud je menší než 25% tak měření ukončí a uloží. Metoda `GetHistogram()` vrací aktuální histogram měření. `GetMobileHistogram()` vrací redukovaný aktuální histogram měření (počet binů je desetinový oproti `GetHistogram`) a metoda `GetDataLog()` vrací aktuální log událostí. Metoda `StopWithoutSave()` zastaví měření bez uložení tím, že zastaví sběr dat a ukončí všechna pomocná vlákna. Metoda `StopWithSave(string notes)` kromě zastavení měření také přes třídu `Common.MeasurementContext` uloží měření s logem událostí a histogramem událostí do SQLite databáze.

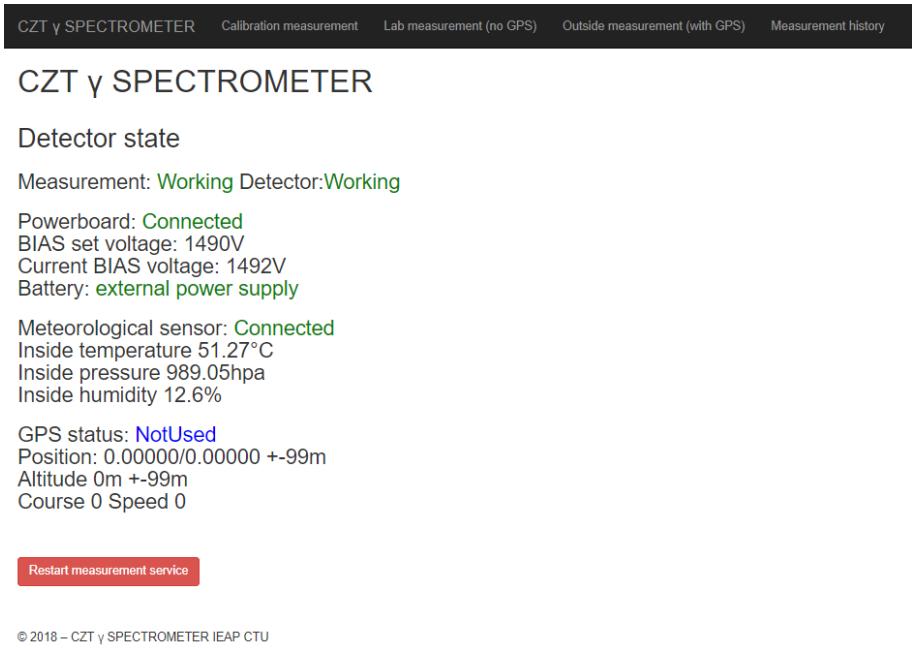
4.3 Webová aplikace

Jako uživatelské rozhraní slouží webová aplikace psaná v C# a ASP.NET MVC 5, která je přístupná po připojení na Wifi vysílanou detektorem a po otevření adresy 192.168.137.1 nebo přes mobilní sít, o čemž pojednává další kapitola. Architektonicky je rozdělena na vrstvy: Vrstva View je tvořena Razor a HTML šablonami, vrstva Modelu, kterou tvoří jednak třída `Common.MeasurementContext` pro přístup k databázi, databázové entity `Common.Model` a také třída `Common.WCFClients` pro komunikaci s měřící službou a vrstva Controlerů, kterou tvoří třídy `HomeControler` (odpovídá za ovládací a sledovací obrazovky měření) a `MeasurementControler`, který umožňuje zobrazit uložená měření. S měřící službou komunikuje webová aplikace pomocí pojmenovaných rour a kontraktů popsaných v předchozí kapitole. Server překládá Ajax požadavky od klienta do formy požadavků přes pojmenovanou rouru příslušné službě měřící služby, která tento požadavek na daném senzoru vykoná. Všechny tlačítka vykonávající prostřednictvím Ajax akci v měřící službě jsou proti `Cross-site Request Forgery` útoku chráněny pomocí `RequestVerificationToken` který je validován na straně serveru systémovou třídou `System.Web.Helpers.AntiForgery`.

Metoda `Status()` na adrese `home/status` vrací JSON objekt, který shrnuje stav detektoru, aby server mohl monitorovat stav všech detektorů. Metoda `RestartMeasurementService()` umožnuje zrestartovat měřící služby pomocí systémové třídy `ServiceControler`. Histogramy jsou vykreslovány javascriptovou verzí fyzikálního frameworku ROOT – JSROOT. Tato knihovna generuje grafy na základě JSON objektů, proto byla naimplementována třída `JSROOTHistogram`, která vezme objekt třídy `Histogram` vrácený měřící službou a převede ho do formátu, kterému

knihovna JSROOT rozumí. Datová nenáročnost aplikace je zajištena minifikací CSS a JS souborů, komprimováním Gzip na úrovni webového serveru, zvýšením obnovovacího intervalu ajaxem z 1s na 5s, a redukcí počtu binů v histogramu.

Na úvodní stránce je přehledová obrazovka (obrázek 4.1) se stavem všech částí měřící služby, která se aktualizuje Ajaxem jednou za sekundu. V dolní části je tlačítko, které restartuje celou měřící službu. Stavy komponent jsou rozlišeny barvami - pokud je komponenta v pořádku je její stav zeleně, pokud má komponenta nějaký problém, tak je její stav červeně a navíc u GPS, pokud není používána, tak je její stav modře. V případě stavu baterie, pokud je napájeno z externího zdroje, nebo stav baterie je větší než 50%, tak je zobrazen zeleně, stav baterie mezi 50-25% je zobrazen žlutě a stav baterie pod 25% červeně.



Obrázek 4.1. Úvodní stránka aplikace

Na stránce Kalibračního měření (obrázek 4.2) se vlevo nachází tabulka s aktuálním stavem všech komponent. Dále je zde možnost nastavit napětí zdroje vysokého napětí, spustit měření a zastavit měření s možností jeho uložení do databáze v detektoru. Vpravo se po zahájení měření vykreslí nezkalibrovaný energetický histogram záření. Pod spektrem se nachází nástroj na výpočet kalibrace, která se provádí tak, že se k detektoru přiloží radioaktivní zdroje produkující záření o známých energiích. Pozice příslušných peaků ve spektru jsou zapsány do formuláře spolu s tabulovanou energií těchto zdrojů a nástroj následně spočítá z lineárního fitu kalibrační parametry A a B rovnice y= A*x+B. Tyto parametry můžeme poté uložit do databáze.

Na stránce Laboratorního měření je vlevo stavová tabulka, ta je stejná jako u Kalibračního měření, je zde navíc formulář pro zadání kalibračních konstant A a B, jejichž hodnoty jsou předvyplněny podle údajů uložených v databázi. Na této stránce se spouští měření s aplikovanou energetickou kalibrací. Spektrum vpravo ukazuje již data zkalibrovaná pomocí těchto konstant.

Na stránce Měření v terénu oproti Laboratornímu měření přibyla tlačítka na zapnutí a vypnutí GPS modulu. V tomto módu jsou k zaznamenávaným částicím přidávány i GPS pozice jejich nálezu.

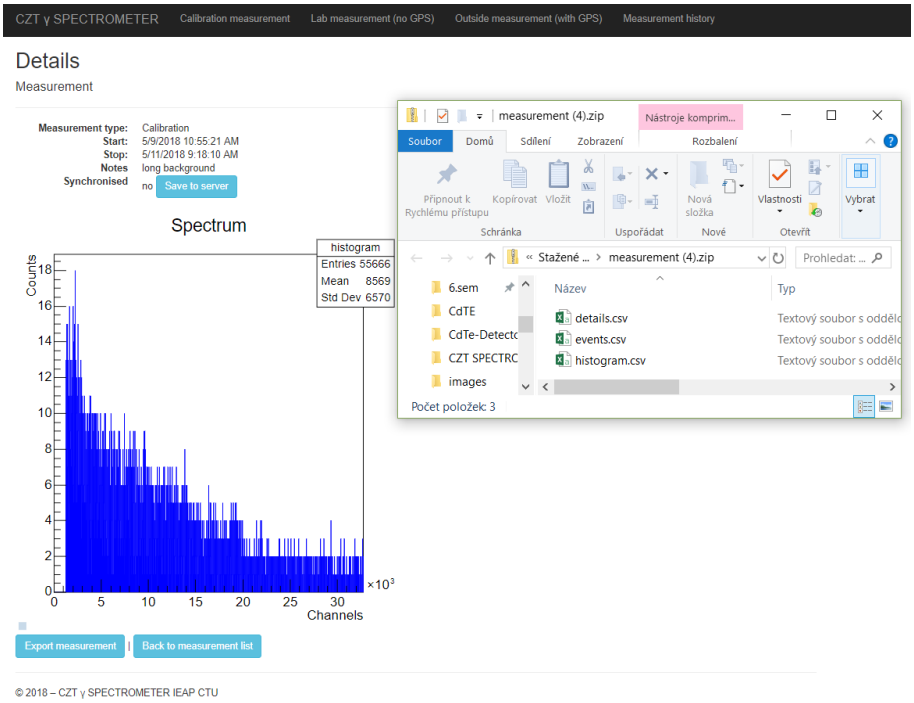
Obrázek 4.2. Obrazovka běžícího kalibračního měření

Na stránce Historie měření jsou vypsány všechny měření uložené v databázi. Měření můžeme smazat nebo si zobrazit detail měření, kde jsou podrobnosti o měření, jeho energetické spektrum, dále možnost stáhnout ZIP archiv obsahující tři CSV soubory s podrobnostmi o měření, histogramem měření a log všech událostí (obrázek 4.3), u terénních měření se navíc zobrazuje mapa se zaznamenanými daty (obrázek 4.4). Pomocí tlačítka `Save to server` je možné odeslat měření na server, to funguje tak, že se měření převedené do JSON formátu knihovnou `Newtonsoft.Json` odešle HTTP požadavkem (pomocí systémové třídy `HttpClient` z detektoru na serveru, který ho uloží.

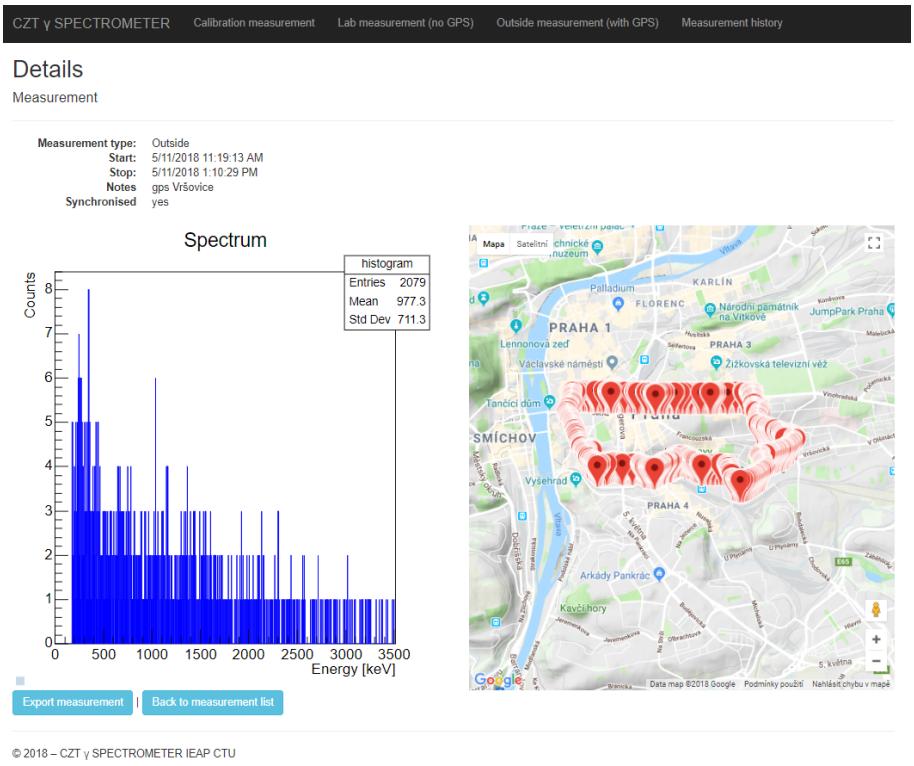
4.4 Seznam použitých knihoven

V softwaru pro detektor byly použity následující knihovny a technologie:

- C#
- ASP.NET MVC 5
- FTDI – knihovna výrobce FTDI čipů pro komunikaci po sériové lince připojenou do USB portu počítače přes FTDI čip [5]
- Yocto-Meteo – knihovna výrobce meteorologického senzoru [3]
- Windows Communication Foundation
- Windows Sensor and Location Platform – lokalizační a polohová vrstva systému
- Bootstrap - CSS framework [6]
- Toastr - JS knihovna pro notifikace [7]
- Json.NET - knihovna pro práci s JSON formátem [8]
- System.Data.SQLite - podpora SQLite pro Entity Framework
- Google Maps [9]
- Apache log4net – logovací knihovna [10]
- JSROOT – javascriptová verze fyzikálního framework ROOT [11]



Obrázek 4.3. Export do ZIP archivu s CSV soubory



Obrázek 4.4. Detail terénního měření na pražském Vyšehradě

Kapitola 5
Implementace softwaru na serveru

Za běžných podmínek má zařízení připojené do mobilní sítě neveřejnou IP adresu (veřejná IP adresa je za poplatek [12]) a je tedy z internetu nepřístupné. Možnosti vzdáleného ovládání detektoru přes mobilní sít bylo dosaženo vytvořením VPN sítě pomocí programu OpenVPN na serveru a reverzní HTTPS proxy pomocí webového serveru Nginx.

5.1 Virtuální privátní sít

OpenVPN na serveru (Debian 7) se nainstaluje příkazem `apt-get install easy-rsa openvpn`. Generování certifikační autority, kterou budou podepsány certifikáty používané v rámci VPN sítě, pomocí programu easy-rsa se zahájí přidáním údajů o certifikátu do vstupního konfiguračního souboru `/etc/openvpn/vars`:

```
export KEY_COUNTRY="CZ"
export KEY_PROVINCE="CZ"
export KEY_CITY="Prague"
export KEY_ORG="IEAP CTU"
export KEY_EMAIL="filip.rychnovsky@utef.cvut.cz"
export KEY_NAME=cdznte.utef.cvut.cz
export KEY_OU=CdTE
```

Dalším krokem je vygenerování parametrů Diffieho–Hellmanovy výměny klíčů tedy prvočíslo p a generátor g pomocí příkazu `./build-dh`. Diffieho–Hellmanova výměna klíčů se používá při budování šifrovaného spojení mezi klientem a serverem. Kořenový certifikát certifikační autority se vytvoří příkazem `./build-ca`. Certifikát pro OpenVPN server se vytvoří příkazem `./build-key-server cdznte`. Konfigurační soubor OpenVPN `cdznte.conf` obsahuje:

```
mode server
tls-server
port 1194
proto tcp-server
dev tap1
client-config-dir ccd

ca /etc/openvpn/easy-rsa/keys/ca.crt
cert /etc/openvpn/easy-rsa/keys/cdznte.crt
key /etc/openvpn/easy-rsa/keys/cdznte.key
dh /etc/openvpn/easy-rsa/keys/dh1024.pem

ifconfig 10.0.0.254 255.255.255.0
ifconfig-pool 10.0.0.1 10.0.0.20 255.255.255.0
ifconfig-pool-persist ipp.txt
```

```
client-to-client

keepalive 10 120
comp-lzo
user nobody
group nogroup
persist-key
persist-tun
status /tmp/openvpn.status 1
log-append /var/log/openvpn.log
status-version 3
verb 4
mute 20
```

Certifikát pro první detektor se vygeneruje pomocí příkazu `./build-key-client detektor-1`. Statická IP adresa 10.0.0.1 v rámci VPN sítě pro první detektor se zajistí vytvořením souboru `/etc/openvpn/ccd/detektor-1` s obsahem `ifconfig-push 10.0.0.1 10.0.0.254`. Konfigurace OpenVPN klienta po jeho instalaci na počítač v detektoru se provede klientským konfiguračním souborem `cdznte.ovpn`.

```
client
remote cdznte.utef.cvut.cz
ca ca.crt
cert detektor-1.crt
key detektor-1.key
remote-cert-tls server
comp-lzo yes
dev tap1
proto tcp
nobind
auth-nocache
script-security 2
persist-key
persist-tun
pull
```

Soubory `ca.crt`, `detektor-1.key` a `detektor-1.crt` zmíněné v konfiguračním souboru pochází ze z složky `/etc/openvpn/easy-rsa/keys/` na serveru. Konfigurace OpenVPN klienta se dokončí nastavením automatického spuštění a připojením do VPN sítě po spuštění počítače.

5.2 Nginx

Na serveru byl nainstalován webový server Nginx a klient certifikační služby Let's Encrypt `cryptbot-auto`, spolu s jazykem PHP a databázovým serverem MySQL (které budou potřeba pro běh serverové webové aplikace, která je popsána v následující sekci). Vzhledem k starší verzi operačního systému (Debian 7) na virtuálním serveru bylo použito PHP 5.6 a MySQL 5.7 z alternativních repositářů (v systémovém repositáři je jen PHP 5.4 a MySQL 5.5).

Nginx plní na serveru několik úloh - první virtuální server obsluhuje požadavky detektorů ve VPN na server (tedy hlavně nahrávání měření), druhý přesměrovává z HTTP na HTTPS a z IP adresy serveru na hlavní doménu, na třetím funguje vlastní serverová

webová aplikace a čtvrtý funguje jako reverzní proxy, kdy požadavky na doménu cdte-1.cdznte.utef.cvut.cz předává do VPN sítě na detektor č.1 (aktuálně je v provozu jeden detektor, další je vyráběn kolegy v ÚTEF) a tomu také odpovídá jeho konfigurační soubor `/etc/nginx/sites-available/default`.

```
# server in VPN
server {
        listen 10.0.0.254:80;
        location / {
                try_files $uri $uri/ /index.php?$args;
        }
        include /etc/nginx/common/nette_conf;
}
# redirect from ip to main server and from http to https
server {
        listen 80 default_server;
        if ($host ~ cdznte\.utef\.cvut\.cz) {
                return 301 https://$host$request_uri;
        }
        return 301 https://cdznte.utef.cvut.cz$request_uri;
}
# public main server
server {
        include /etc/nginx/common/ssl_conf;
        server_name cdznte.utef.cvut.cz;
        include /etc/nginx/common/nette_conf;
        location / {
                try_files $uri $uri/ /index.php?$args;
                include /etc/nginx/common/basic_auth;
        }

}
# proxy for detector n.1
server {
        include /etc/nginx/common/ssl_conf;
        server_name cdte-1.cdznte.utef.cvut.cz;
        location / {
                proxy_pass http://10.0.0.1;
                include /etc/nginx/common/basic_auth;
        }
}
```

Aby se společné sekce neopakovali, tak byly vyděleny do samostatných souborů. Prvním je `/etc/nginx/common/ssl_conf`, který obsahuje konfiguraci SSL certifikátu Let's Encrypt, který byl nainstalován, a je každé 3 měsíce automaticky obnovován pomocí nástroje `cerbot-auto`:

```
listen 443 ssl
ssl_certificate /etc/letsencrypt/live/cdznte.utef.cvut.cz/fullchain.pem
ssl_certificate_key /etc/letsencrypt/live/cdznte.utef.cvut.cz/privkey.pem
include /etc/letsencrypt/options-ssl-nginx.conf
ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem
```

Dalším společným souborem je `/etc/nginx/common/nette_conf`, který specifikuje document root a zpracování PHP souborů PHP enginem:

```
# Nginx config for nette
root /var/www/www;
index index.php;
# pass the PHP scripts to FPM
location ~ \.php$ {
        fastcgi_pass unix:/var/run/php5-fpm.sock;
        fastcgi_index index.php;
        include fastcgi_params;
}
# deny access to .htaccess files
location ~ /\.ht {
        deny all;
}
```

Posledním je krátký `/etc/nginx/common/auth_conf`, který specifikuje zabezpečení pomocí ověření HTTPS Basic.

```
auth_basic "CdTe server";
auth_basic_user_file /etc/nginx/.htpasswd;
```

5.3 Webová serverová aplikace

Webové rozhraní má Model-View-Controler architekturu, závislosti třídám jsou dodávány pomocí Depedency Injection.

5.3.1 Model

Vrstva modelu se skládá z Entit, tedy tříd `BaseEntity`, `Measurement`, `Event`, `Histogram`, `Detector`, `DetectorState` a `DetectorEvent` (popsány v diagramu na obrázku 3.2), Data object access (DAO) tříd, které pomocí Object-relational mapping knihovny Doctrine přistupují k databázi a poskytují metody pro manipulaci s databází a Servisních tříd, v kterých je buisness logika.

`BaseDao` v konstruktoru z `\Kdyby\Dotcrine\EntityManager` vytvoří `\Kdyby\Doctriney\EntityRepository` pro příslušnou Entitu která je specifikována v konstruktoru odvozených DAO tříd pro jednotlivé entity. Metoda `findAll()` vrací pole všech entity daného typu z databáze. Metoda `findById(int $id)` vrací entitu se zadaným id, pokud taková existuje. Metoda `findBy(array $criteria)` vrací entity, které odpovídají zadaným podmínkám. Metoda `findOneBy(array $criteria)` je podobná, jen místo všech vyhovujících, vrací první vyhovující entitu. Poslední metoda `persist(AbstractEntity $entity)` uloží entitu do databáze.

Většina DAO tříd tedy `DetectorDao`, `DetectorStateDao`, `MeasurementDao`, `MeasurementTypeDao` a `EventDao` jen ve svém konstruktoru volají konstruktor předka tedy `BaseDao` s označením třídy. Vyjímkou je `DetectorEventDao`, která obsahuje metodu `findEventsByDetectorId($id)`, která vrací události detektoru s podporou stránkování.

Servisní vrstva modelové vrstvy se skládá z tříd `DetectorService`, `EventService`, `DetectorEventService` a `MeasurementService` a pomocných `JSROOTHistogram`, `MeasurementJson`, `DetectorStateService`,`MeasurementTypeService`.

Třída `DetectorService` poskytuje služby kolem třídy `Detector`. V konstruktoru jsou vloženy třídy `DetectorDao` a `DetectorStateService`. Metody `findAll()`, `findById($id)`, volají stejnojmenné výše zmíněné metody třídy `DetectorDao`. Metoda

`findByIP($ip)` vrací objekt detektoru podle ip adresy v rámci VPN sítě. Nejrozsáhlejší metodou je `updateDetectorsStatus()`, která z databáze vytáhne seznam detektorů a každého se dotáže na jeho stav vysláním HTTP GET požadavku na adresu `http://$detector_ip/home/status` pomocí knihovny `Guzzle`. Webový server detektoru na tento požadavek odpoví odpoví ve formátu JSON objektu s dvěma atributy, prvním je stav detektoru (možné hodnoty `ready`, `measuring`, `offline` nebo `error`) a druhým je zpráva s podrobnostmi. Po příjmu je odpověď ve formátu JSON zparsována třídou `Json` z `Nette Frameworku`, následně je pomocí knihovny `League/JsonGuard` ověřena validnost odpovědi vůči schématu, tedy, že obsahuje oba výše zmíněné atributy a že stav detektoru je mezi vyjmenovanými položkami. Následně je porovnán poslední známý a aktuální stav detektoru a do historie detektoru je zapsána příslušná událost. Tato metoda samozřejmě počítá s tím, že detektor nemusí být dostupný, nebo že může vrátit neplatnou odpověď a i toto pozná a do historie zapíše.

Třída `DectorEventService` má kromě konstruktoru, ve kterém je vložena třída `DetectorEventDao`, jen jednu metodu a to `findEventsByDetectorId($id)`, která voláním stejnojmenné metody z `DetectorEventDao` vrací události detektoru s podporou stránkování.

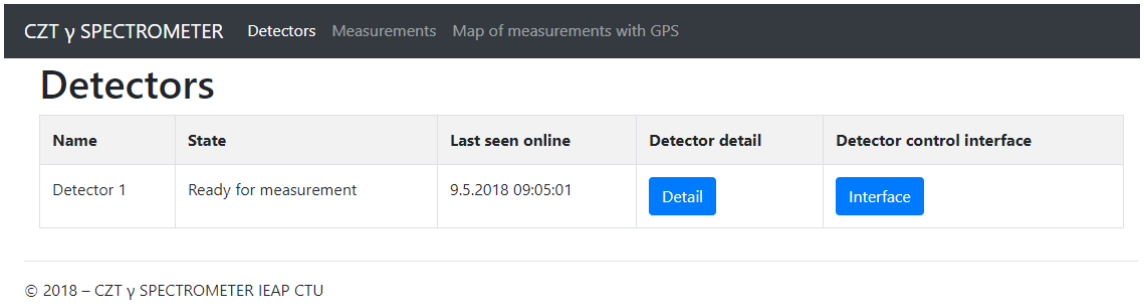
Třída `EventService` má kromě konstruktoru, kde jsou vloženy třídy `EventDao` a `MeasurementTypeService` jen jednu metodu a to `getGPSMeasurementEvents()`, která vrací pole naměřeného záření ze všech měření provedených ve venkovním modu, tedy se zapnutou GPS.

Třída `MeasurementService` poskytuje služby kolem ukládaní a exportování měření. Kromě toho obsahuje metodu `findById($id)`, která vrací `Measurement` podle id. Metoda `save(Detector $detector, $body)` se stará o uložení měření, které uživatel odeslal z detektoru na server. Nejprve je třídou `MeasurementJson` zajištěno parsovaní a validace JSON, které detektor odeslal. Následně jsou validovaná data uložena do entity `Measurement` s polem objektů třídy `Event` (což je log detekovaného záření) a `Histogram`. Poté je měření připojeno k objektu detektoru a uloženo v databázi. Další metodou je `exportToZip($measurement_id)`, která poskytuje export měření v podobě zip archivu s třemi CSV soubory: measurement.csv, histogram.csv a events.csv. Metoda nejdříve pomocí knihovny `League/Csv` sestaví zmíněné CSV soubory a poté je pomocí třídy ZipArchive složí do jednoho ZIP archivu.

Třída `JSROOTHistogram` má jedinou veřejnou statickou metodu `fromHistogram(Histogram $histogram)`, která převede objekt Histogramu z databáze do podoby, kterému po převedení do JSON formátu rozumí knihovna JSROOT, která se používá na vykreslení histogramů. K tomuto používá neveřejné metody `getAxis($name, $title, $bins, $min, $max)`, která vypisuje objekt os, `getFfunctions($histo)`, která za pomoci `getPaveStats($histo)`, `getPaveStatsLines($histo)` a `getStatLine($title, $textSize, $originSize)` vypíše legendu histogramu.

Třída `DetectorStateService` vrací z databáze za pomocí třídy `DetectorSateDao` všechny možné stavy detektoru to: `getReadyState()`, `getMeasuringState()`, `getErrorState()`, `getOfflineState()` a `getStateByName($type)`, která vrací objekt stavu podle jména.

Třída `MeasurementTypeService` vrací z databáze za pomocí třídy `MeasurementTypeDao` všechny typy měření a to `getNoneState()`, `getInsideState()`, `getCalibrationState()`, `getOutsideState()` a `getTypeByName($type)`, která vrací objekt typu měření dle jména.



Obrázek 5.1. Server - Hlavní strana

5.3.2 Vrstvy kontrolérů a šablon

Dalšími dvěma vrstvami jsou vrstvy Controlerů a View (ve kterém se používají Latte šablony), vzhledem k tomu, že tyto dvě vrstvy jsou úzce spjaty, jsou popsány společně.

První kontroler je `DetectorPresenter`, který obsahuje dvě stránky - hlavní se seznamem detektorů (obrázek 5.1), který je získán z databáze pomocí metody `findAll()` třídy `DetectorService` a detail detektoru s jeho měřeními a historií stavů, který je stránkovatelný pomocí knihovny `VisualPaginator` (obrázek 5.2).

Dalším je `MeasurementPresenter`, který zajišťuje jednak zobrazení detailu měření s histogramem za pomocí výše zmíněné `JSROOTHistogram` a u venkovního měření s Google mapou, na které jsou zobrazeny pozice uloženého záření (obrázek 5.3). Zobrazuje také seznam všech měření s možností měření smazat, mazací odkaz je proti Cross-Site Request Forgery (CSRF) chráněn pomocí knihovny Nextras Secured Links, která přidává do odkazu jedinečný token (hash celé url a id uživatelské session pomocí HMAC metody s náhodným klíčem jedinečným pro každého uživatele). Dále zajišťuje za pomoci výše zmíněné metody `exportToZip($id)` export měření v podobě ZIP archivu s CSV soubory. Dále poskytuje rozhraní pro ukládání měření z detektorů, kdy v metodě `actionSave()` nejprve zkontroluje zda požadavek je typu `POST` a že přišel z VPN sítě od existujícího detektoru podle IP adresy rozhraní serveru a detektoru. Následně je metodou `save(Decteor $dector, $body)` třídy `MeasurementService` měření uloženo.

Posledním je `MapPresenter`, který má za úkol vykreslit Google mapu se všemi událostmi, k čemuž používá `getGPSMeasurementEvents()` třídy `EventService` (obrázek 5.4).

Trochu stranou stojí třída `DetectorStatusCommand`, která na příkaz (pomocí crontabu uživatele www-data spouštěno každých 5 sekund):

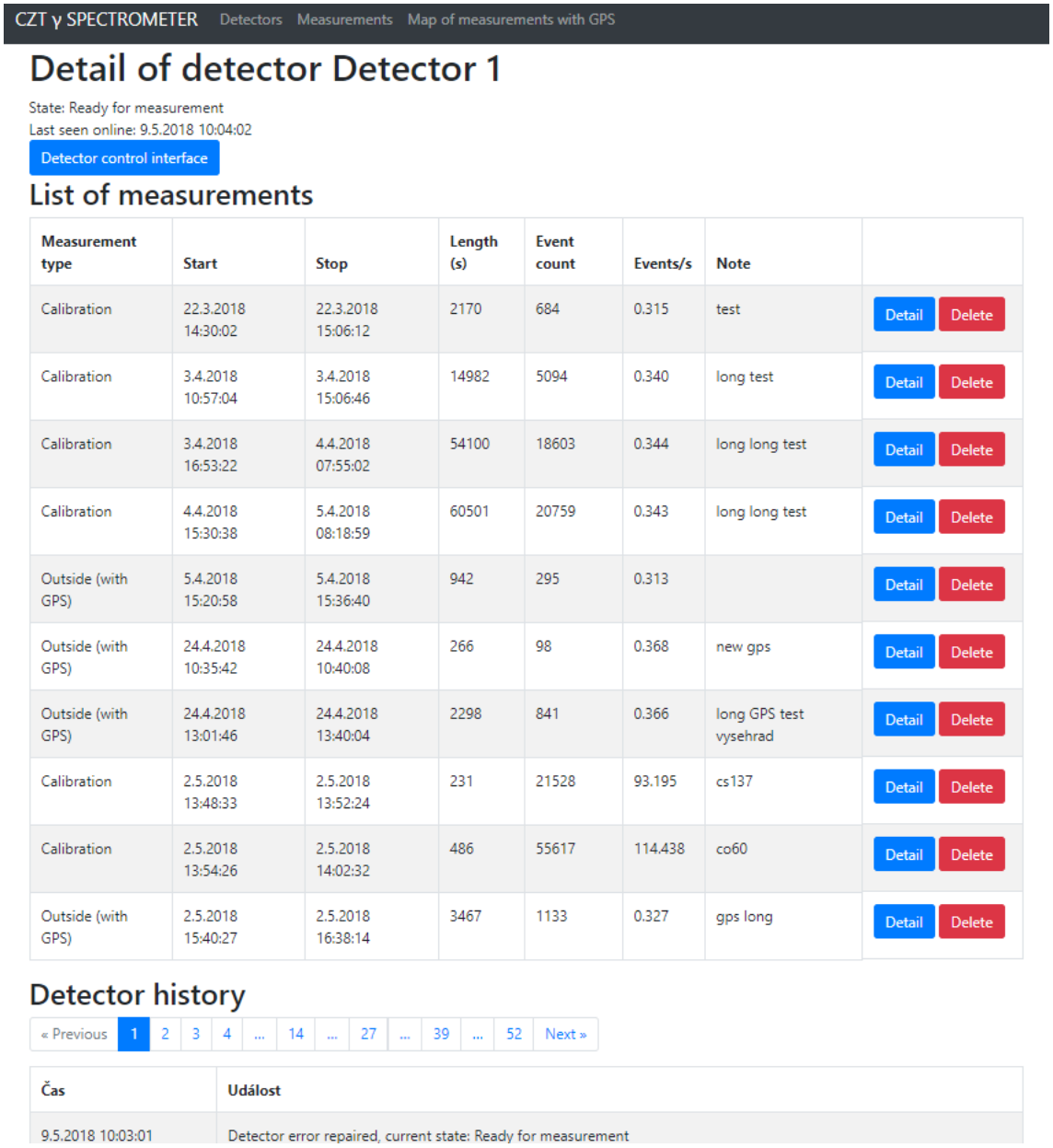
```
cd /var/www
php ./www/index.php app:getDetectorStatus;
```

Zavoláním výše zmíněné metody `updateDetectorsStatus()` třídy `DetectorService` aktualizuje stav všech detektorů.

5.4 Seznam použitých knihoven

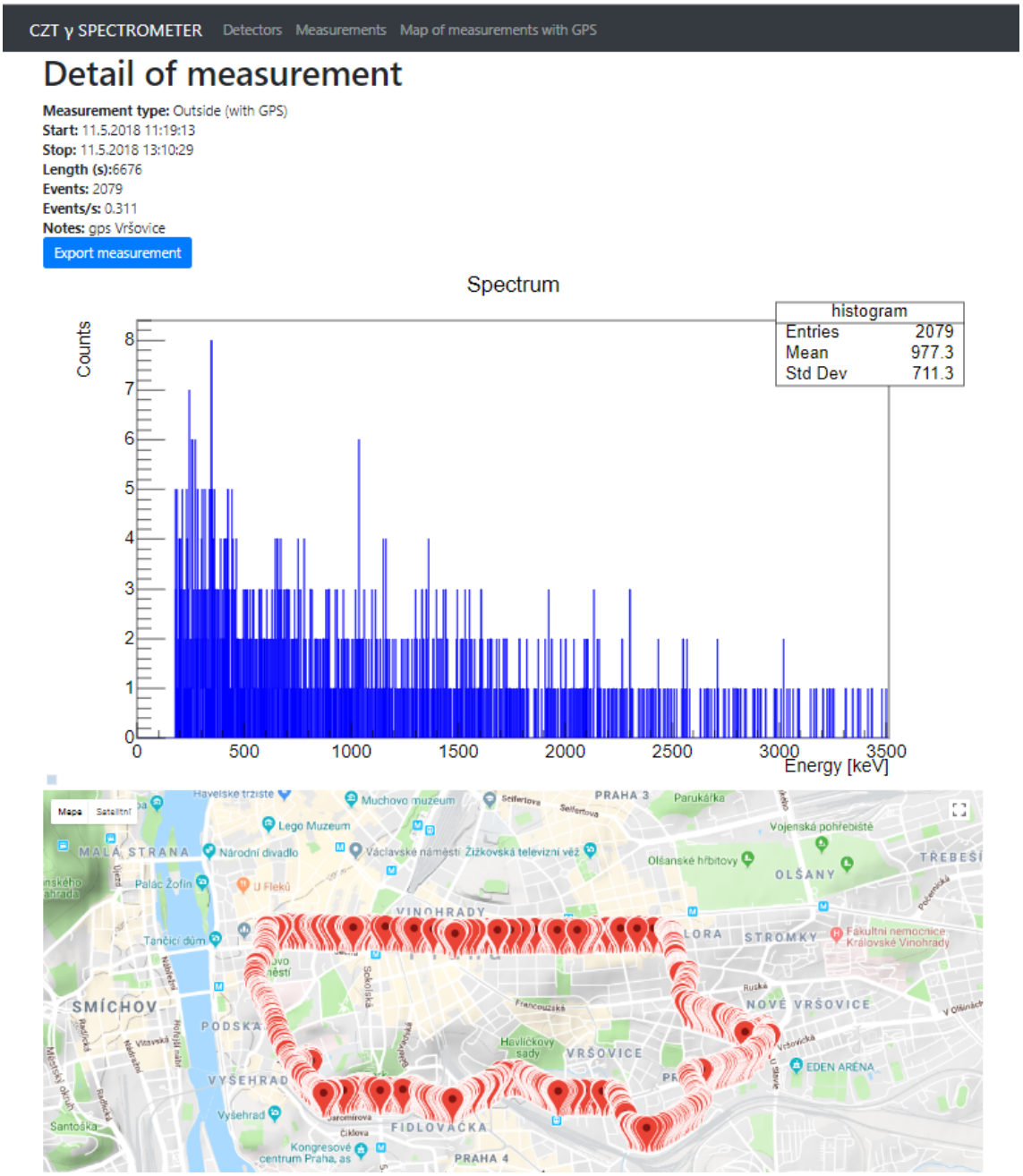
Na serveru byly použity následující knihovny a technologie:

- PHP
- Nginx
- OpenVPN [13]
- MySQL

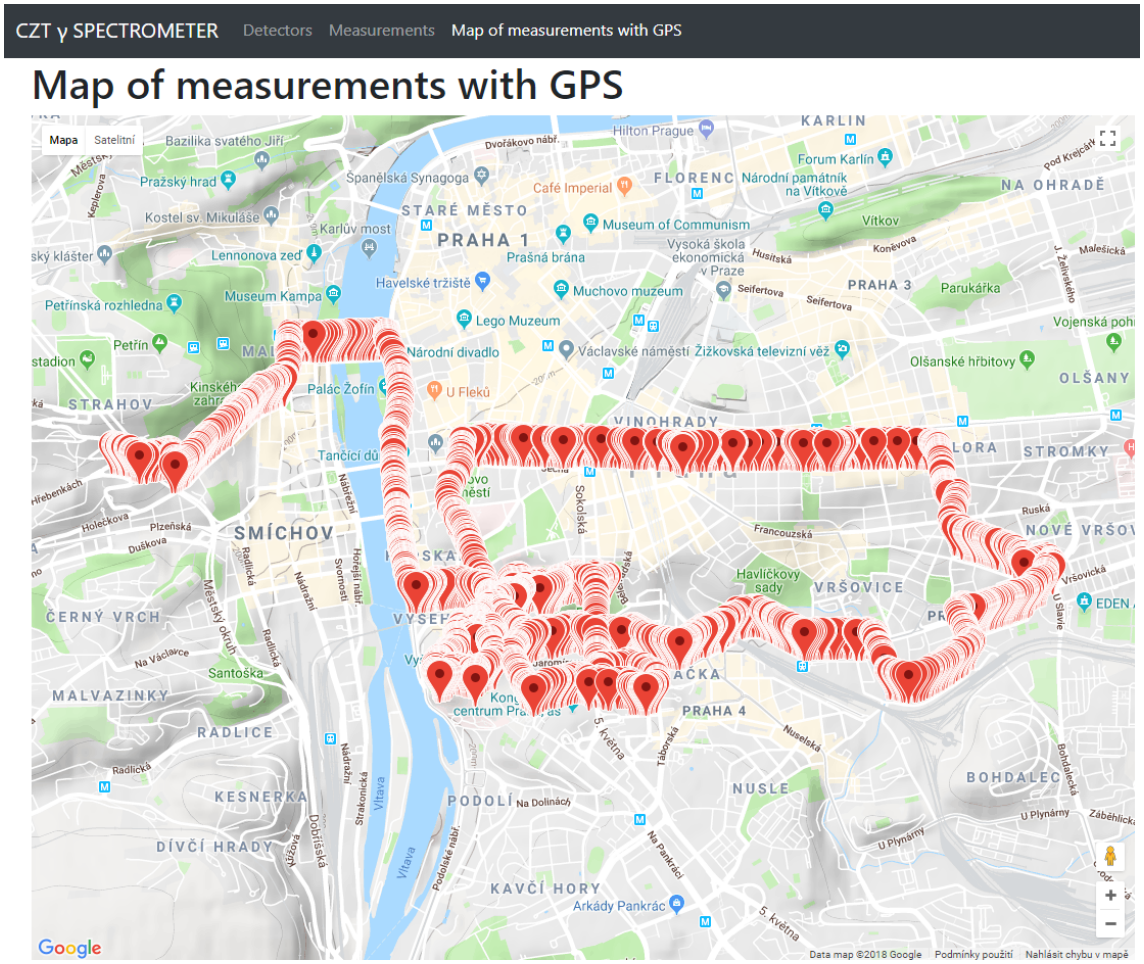


Obrázek 5.2. Server - Detail detektoru

- Doctrine - ORM knihovna pro přístup k databázi [14]
- Nette Framework - český PHP MVC framework [15]
- Nextras Secured Links - doplněk do Nette Framorku, ochrana odkazů proti CSRF [16]
- Kdyby Doctrine - wraper Doctrine pro Nette framework [17]
- Guzzle - knihovna pro vytváření HTTP požadavků v PHP [18]
- League JSON guard - knihovna na validaci JSON podle zadaného schématu [19]
- League CSV - knihovna pro práci s CSV formátem [20]
- Visual Paginator - knihovna pro stránkování [21]
- Boostrap - CSS framework [6]
- Google Maps [9]
- JSROOT – javascriptová verze fyzikálního framework ROOT [11]



Obrázek 5.3. Server - Detail měření

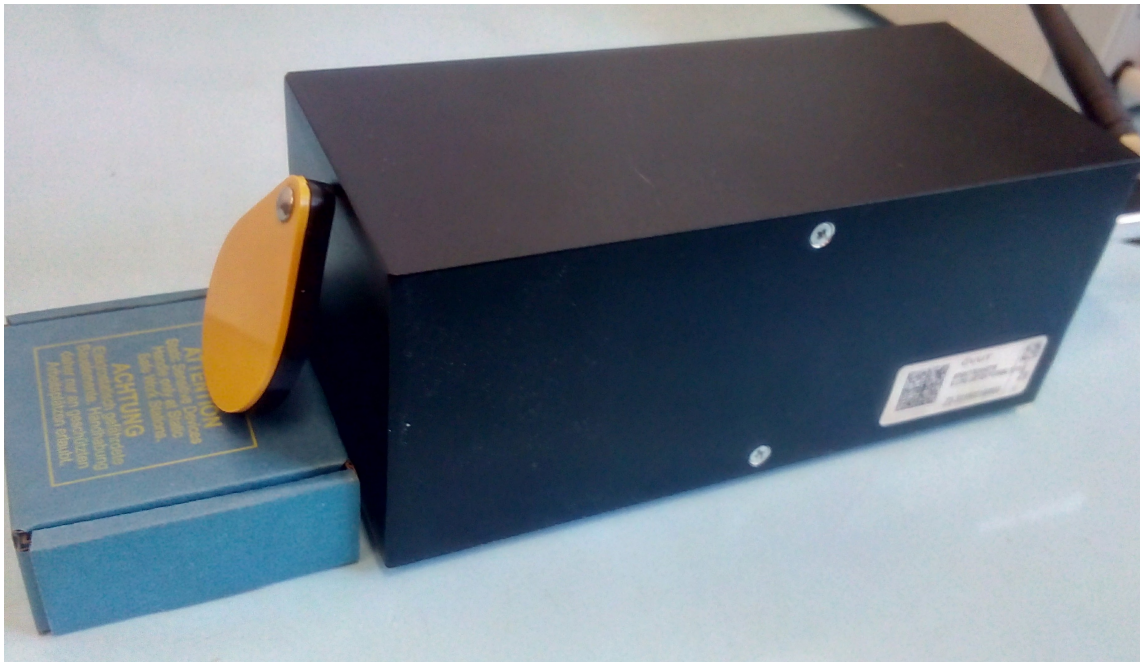


Obrázek 5.4. Server -Mapa měření s GPS

Kapitola 6
Testování softwaru

Nejprve bylo otestováno vyčítání údajů z jednotlivých senzorů porovnáním hodnoty s externě zjištěnou hodnotou. U `PowerBoardService` bylo čtení stavu baterie ověřeno porovnáním s hodnotou zjištěnou multimetrem (zjištěn rozdíl 0,2 V při napětí 10,8 V), operace se zdrojem vysokého napětí byly porovnány vzájemně, tedy že bylo vysoké napětí nastaveno na 1500V a vůči této hodnotě byly porovnány zpětné vazby ze zdroje vysokého napětí (nastavená a aktuální hodnota), které kolísají kolem 1490V. U `GPSService` byla zjištěná poloha zkontrolována pomocí mapy. U `MeteoService` byla zkontrolována teplota pomocí senzorů integrovaných na základní desce počítače, které byly vyčteny pomocí programu HWMonitor od společnosti CPUID.

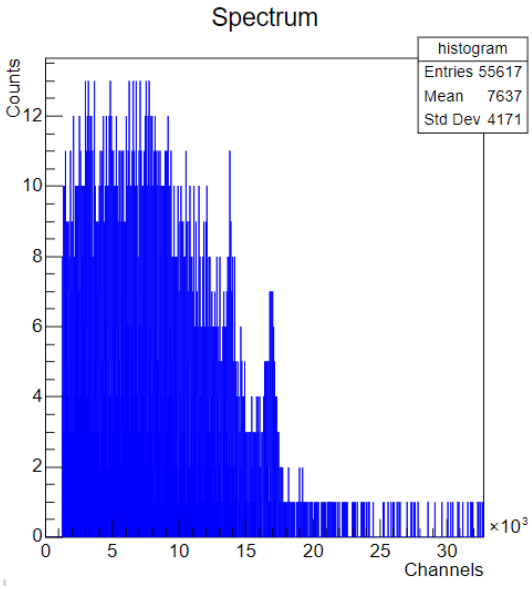
Po otestování senzorů byla provedena kalibrace CdZnTe krystalu přiložením (obrázek 6.1) zdrojů ionizujícího záření ^{60}Co a ^{137}Cs (obrázek 6.2). Ze spekter ^{60}Co (obrázek 6.3) a ^{137}Cs (obrázek 6.4) byly nalezeny peaky ^{137}Cs 661keV na pozici 3851 kanálu, peaky ^{60}Co 1173keV a 1322keV na pozicích 13800 a 16820 kanálů. Vzhledem k tomu, že u těchto měření během sekundy přilétlo průměrně 100 částic, tak toto měření posloužilo i jako test rychlosti a robustnosti celého sytému. Dále byla ověřena dlouhodobá stabilita 47h měřením (obrázek 6.5) . Dále bylo provedeno několik venkovních měření v ulicích Prahy (obrázek 5.4).



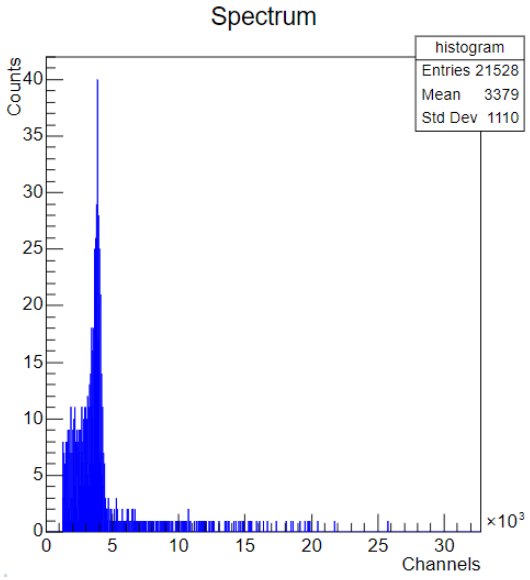
Obrázek 6.1. Kalibrace detektoru zdrojem ionizujího záření (v plastovém obalu)

Na schůzi Neutrinové skupiny Oddělení fyzikálních experimentů ÚTEF 24.4.2018 bylo prezentováno uživatelské rozhraní aplikace. Z diskuze vzešel již realizovaný požadavek na převod uživatelského rozhraní do angličtiny.

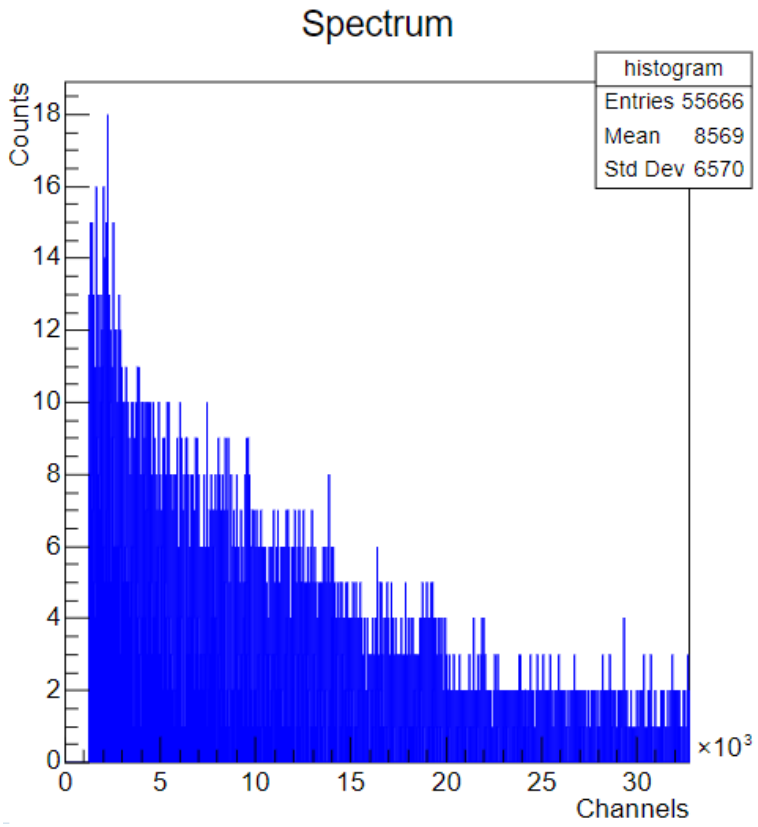
Obrázek 6.2. Kalibrační zdroje ionizujícího záření ^{60}Co a ^{137}Cs



Obrázek 6.3. Spektrum ^{60}Co s peaky 1173keV a 1322keV na pozících 13800 a 16820 kanálů



Obrázek 6.4. Spektrum ^{137}Cs s 661keV s peakem na pozici 3851 kanálu



Obrázek 6.5. Spektrum dlouhodobého pozaďového měření (47h)

Kapitola 7
Závěr

Cílem této bakalářské práce bylo především navrhnout a implementovat ovládací software pro detektor ionizující záření vyvinutého v ÚTEF ČVUT v Praze, na bázi polovodiče CdZnTe. Vyvinutý software umí komunikovat se všemi komponentami tzn. s FastADC převodníkem přes sériovou linku, s mikrokontrolerem desky zdrojů přes sériovou linku, Meteorologickým senzorem pomocí knihovny výrobce a GPS senzorem pomocí lokalizačních služeb systému Windows. Jednotlivé části softwaru jsou rozděleny do malých celků, které spolu komunikují pomocí pojmenovaných rour. To zajišťuje odolnost proti pádu a spolehlivost aplikace, která je posílena pečlivým ošetřováním a řešením výjimek.

Vyvinuté uživatelské rozhraní v podobě webové aplikace přístupné přes Wifi umožňuje zobrazovat stav všech částí zařízení, restartovat měřící službu, řídit napětí zdroje vysokého napětí potřebného pro aktivaci CdZnTe krystalu, umožňuje nastartovat měření v celkem třech módech – kalibračním, laboratorním (oproti kalibračnímu módu aplikována kalibrace) a terénním (oproti laboratornímu módu aktivována GPS). Během měření webové rozhraní zobrazuje energetické spektrum záření. Při ukončení měření tlačítkem Stop lze měření uložit. Aplikace umožňuje zobrazení seznamu uložených měření a detaily měření s podrobnostmi, energetickým spektrem záření a u terénních měření mapu detekovaného záření. Dále je zde možnost stáhnout ZIP archiv obsahující tři CSV soubory s podrobnostmi o měření, histogramem měření a log všech událostí. Uložené měření lze odeslat na server.

K detektorům lze přistupovat vzdáleně přes mobilní sít pomocí nakonfigurované OpenVPN sítě a Nginx, který slouží jako reverzní proxy. Na virtuálním serveru v clusteru ÚTEF, kde byla nakonfigurována OpenVPN sít a Nginx, běží serverová webová aplikace umožňující zobrazení stavu všech detektorů, zobrazení detailu detektoru s historií jeho stavů a provedenými měření, zobrazení detailu měření s energetickým spektrem a případnou mapu. Dále je zde možnost stáhnutí měření v ZIP archivu. Dále umožňuje zobrazit mapu detekovaného záření ze všech provedených měření s GPS.

Datová náročnost webového rozhraní detektoru byla snížena minifikací souborů, kompresí na úrovni webového serveru a VPN sítě, snížením počtu binů v histogramech a možností ukládání měření na server, čímž se potřeba přístupu na detektor výrazně omezí.

7.1 Další rozvoj aplikace

V připravovaném druhém detektoru je kolegy plánováno přidání LED diody na předním panelu detektoru jako indikace stavu detektoru a možnost vypnutí GSM modemu. Aplikace bude proto dále rozšířena, aby tento přidaný hardware podporovala.

Literatura

[1] Glenn F. Knoll. *Radiation detection and measurement*. 4th ed. vydání. Hoboken, N.J.: John Wiley, 2010. ISBN 04-701-3148-9.

[2] Christian Oldorf. *Operation of CdZnTe Semiconductor Detectors in Liquid Scintillator for the COBRA Experiment*. 2015.

[3] *Yocto-Meteo, User's guide*. 2017.
http://www.yoctopuce.com/projects/yoctometeo/METEOMK1.usermanual-EN.pdf.

[4] *U-blox GNSS Sensor and VCP Device Driver - User Guide*. 2016.
https://www.u-blox.com/sites/default/files/products/documents/u-blox-GNSS-Sensor-and-VCP-Device-Driver_UserGuide_%28UBX-15022397%29.pdf.

[5] *FTD2XX.Net*. 2014.
http://www.ftdichip.com/Support/SoftwareExamples/CodeExamples/CSharp.htm.

[6] *Bootstrap*. 2018.
https://getbootstrap.com.

[7] *Toastr by CodeSeven*. 2018.
http://codeseven.github.io/toastr/.

[8] *Json.NET - Newtonsoft*. 2018.
https://newtonsoft.com/json.

[9] *Google Maps JavaScript API*. 2018.
https://developers.google.com/maps/documentation/javascript/adding-a-google-map.

[10] *Apache log4net*. 2017.
http://logging.apache.org/log4net/.

[11] *JavaScript ROOT*. 2017.
https://root.cern.ch/js/.

[12] *CENÍK TARIFŮ A SLUŽEB PRO TARIFNÍ A TWIST ZÁKAZNÍKY T-MOBILE PLATNÝ K 16. 4. 2018*. 2018.
https://t-mobile.cz/dcpublic/Cenik_sluzeb_T-Mobile-II.pdf.

[13] *OpenVPN Server*. 2018.
https://kb.vpsfree.cz/navody/server/openvpn.

[14] *Doctrine 2 ORM*. 2018.
https://doctrine-project.org/projects/doctrine-orm/en/2.6/index.html.

[15] *Nette framework*. 2018.
https://nette.org.

[16] *Nextras Secured Links*. 2017.
https://nextras.org/secured-links/docs/master/.

[17] Filip Procházka. *Refaktorizace balíčků Kdyby*. 2017.

[18] *Guzzle, PHP HTTP client*. 2018.
http://docs.guzzlephp.org/en/stable/.

[19] *JSON Guard*. 2018.
http://json-guard.thephpleague.com/.

[20] *League Csv*. 2018.
http://csv.thephpleague.com/.

[21] *Visual paginator for Nette Framework*. 2017.
https://github.com/iPublikuj/visual-paginator.

Příloha A
Obsah CD

- `document` Zdrojové tex soubory tohoto dokumentu
- `detector/source` Zdrojové kódy vytvořené aplikace pro detektor
- `detector/compiled/web` Instalační balíček webového rozhraní detektoru
- `detector/compiled/service` Zkompilovaná měřící služba
- `server/source` Zdrojové kódy vytvořené aplikace pro server

Příloha B
Instalace softwaru na detektor

1. Připojit se na Wifi detektoru `CdTe#NtworkAccess` a následně pomocí Vzdálené plochy do počítače dektektoru (192.168.137.1)
2. Nastavit mobilní sít na automatické připojování
3. Nainstalovat Internet Information Services s doplňky HTTP Compression Dynamic a Static Content Compression
4. Do složky `C:/inetpub/MeasurementService` zkopírovat obsah složky `detector/compiled/MeasurementService`
5. Příkazem `C:/Windows/Microsoft.NET/Framework/v4.0.30319/InstallUtil.exe C:inetpub/MeasurementService/CZT_SPECTROMETER.MeasurementService.exe` nainstalovat měřící službu jako Windows službu
6. Pomocí bat souboru `CZT_SPECTROMETER.deploy.cmd` ve složce `detector/compiled/web` naistalovat webové rozhraní detektoru
7. Na serveru `cdznte.utef.cvut.cz` vygenerovat příkazem `./build-key-client detektor-N` (N je pořadové číslo detektoru) ve složce `/etc/openvpn/easy-rsa` klientský certifikát
8. Zajistit statickou IP adresu pro přidávaný detektor pomocí vytvoření souboru `/etc/openvpn/ccd/detektor-N` s obsahem `ifconfig-push 10.0.0.N 10.0.0.254`
9. Nastavit reverzní proxy tzn. v souboru `/etc/nginx/sites-available/default` zkopírovat poslední sekci a upravit IP adresu a doménové jméno. Následně restartovat Nginx
10. Vygenerovat SSL certifikát pro novou reverzní proxy pomocí příkazu `certbot-auto`
11. Na detektoru nainstalovat OpenVPN, nakopírovat certifikáty z předešlých kroků a umístit konfigurační soubor `cdznte.ovpn` (viz kapitola 5.1), nastavit OpenVPN na automatické připojení po naběhnutí OS
12. Přidat detektor na server přidáním řádku do tabulky detector (detector_state_id = 1, name = `název detektoru`, ip = `10.0.0.N`, proxy_path = `https://cdte-N.cdznte.utef.cvut.cz`) nástrojem Adminer na adrese `https://cdznte.utef.cvut.cz/adminer/`