VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2017 **Jan Mitoraj**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jan Mitoraj**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Railsformers s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Seidl, Ph.D.**

Konzultant bakalářské práce: Ing. Jiří Kubica

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

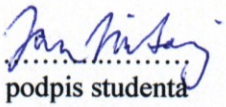
VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky 1D16 OSTRAVA

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne: *25. dubna 2017*



podpis studenta

Poděkování

Rád bych poděkoval Ing. Davidu Seidlovi, Ph.D. za konzultace a připomínky při vytváření této bakalářské práce. Dále bych chtěl poděkovat Ing. Jiřímu Kubicovi a Ing. Davidu Garaiovi, jednatelům společnosti Railsformers s.r.o. za možnost absolvování individuální odborné praxe. Ing. Jiřímu Kubicovi děkuji také za konzultace a pomoc při vypracování.

Prohlášení zástupce spolupracující právnické nebo fyzické osoby

„Souhlasím se zveřejněním této bakalářské/diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských/magisterských programech VŠB-TU Ostrava."

Railsformers s.r.o.
IČ: 24704440 DIČ: CZ24704440
www.railsformers.com
info@railsformers.com

Dne: *25. dubna 2017*

.....
podpis zástupce

Abstrakt

Bakalářská práce popisuje absolvování individuální odborné praxe ve společnosti Railsformers s.r.o. Odborná praxe byla zaměřena na oblast vývoje aplikace pro Internet věcí (IoT). Aplikace v maximální možné míře využívá otevřené technologie, operační systém Linux, platformu Docker a jazyk Ruby.

Cílem praxe je vývoj prototypu brány mezi různými technologiemi, využívanými v Internetu věcí. Součástí praxe je výběr vhodné hardwarové platformy, výběr vhodného operačního systému, tvorba základního aplikačního rozhraní pro správu platformy a implementace aplikace obsluhující samotnou bránu.

Klíčová slova

Bakalářská práce, odborná praxe, Docker, Internet věcí, Railsformers s.r.o, Linux, Ruby

Abstract

This Bachelor's thesis describes a professional practice at the company Railsformers s.r.o. The professional practice has been focused on development of application for Internet of Things (IoT). Application maximizes usage of open technologies, Linux operating systém, Docker platform and Ruby language.

The main goal of professional practice is development of IoT gateway prototype for connection between different interfaces. Parts of practice are: selection of hardware platform, selection of operating systém, creation of basic application interface for maintenance and implementation of application for gateway itself.

Key words

Bachelor's thesis, professional practice, Docker, Internet of Things, Railsformers s.r.o., Linux, Ruby

Obsah

Seznam použitých zkratek

Zkratka	Význam
API	Application Programming Interface
ARM	Advanced RISC Machine
CPU	Central Processor Unit
CSR	Certificate Signing Request
DHCP	Dynamic Host Control Protocol
HTTP	Hypertext Transfer Protocol
I2C	Inter-integrated Circuit
IEEE	Institute of Electrical and Electronic Engineers
IoT	Internet of Things
M2M	Machine to machine
RAM	Random Access Memory
REST	Representional State Transfer
S.M.A.R.T	Self-Monitoring, Analysis and Reporting Technology
SLAAC	Stateless Address Autoconfiguration
SPI	Seriál Peripheral Interface
SQL	Structured Query Language
UART	Universal Asynchronnous Receiver/Transmitter

Seznam ilustrací a seznam tabulek

Úvod

Jako téma vypracování bakalářské práce jsem si zvolil možnost absolvování individuální odborné praxe ve společnosti Railsformers s.r.o. Důvodem tohoto rozhodnutí byl záměr získat reálné pracovní zkušenosti a možnost naučit se pracovat se skriptovacím jazykem Ruby a celým ekosystémem, postaveným kolem tohoto jazyka.

Jak již bylo uvedeno, odbornou praxi jsem vykonal u společnosti Railsformers s.r.o., u které jsem začal pracovat již v srpnu roku 2016 jako DevOps. V úvodní části práce jsou uvedeny základní informace o společnosti, dále je také uveden popis mé pracovní náplně. V dalších částech rozebírám zadané úkoly a postup jednotlivých řešení společně s důvody výběru konkrétního řešení.

Cílem odborné praxe bylo navhnout koncept řešení sběru dat z bezdrátových senzorů tak, aby jednotlivá čidla nebyla závislá na síti provozované třetí stranou a zároveň vytvořit rozhraní pro přístup subjektů třetí strany k naměřeným datům.

1 Popis odborného zaměření firmy

1.1 Popis společnosti

Společnost Railsformers s.r.o. vznikla v roce 2010 oddělením vývoje aplikací ve frameworku Ruby on Rails od vývoje internetových obchodů ve společnosti Skvělý.CZ s.r.o. Společnost se specializuje na vývoj středních a velkých informačních systémů, sociálních sítí a komunitních portálů. Mezi největší projekty patří systém Hedurio, služící jako informační systém pro bezpečnostní agentury. S rozšířením chytrých telefonů musela společnost reagovat na požadavky zákazníků, proto vzniklo vyhrazené oddělení pro vývoj mobilních aplikací pro platformy Android a iOS. Probíhají také testy platformy Ionic pro vývoj hybridnách aplikací pro více platforem zároveň.

Díky rozmachu světa Internetu věcí a projektů jako Smart cities, Smart grid a podobných se společnost ve spolupráci se společností Intel zapojila do vývoje systémů, založených na technologiích LTE-A, LoRa, Sigfox a IQRF®. V kombinaci s již existujícím systémem Hedurio pak tato technologie může být použita například pro hlídání dětí nebo seniorů, případně pro hlídání majetku.

1.2 Popis odborného zaměření studenta

Ve firmě jsem pracoval jako DevOps. Náplní této pozice je převzít aplikace od vývojářů a implementovat je do produkčního prostředí, škálovat je a udržovat v chodu, včetně řešení poruch a různých migrací. V dnešní době je velmi populární také přechod z fyzických serverů do cloudové infrastruktury, několik těchto přesunů jsem také zrealizoval. Měl jsem možnost vyzkoušet skriptovací jazyk Ruby, MySQL a PostgreSQL databáze, fulltextový vyhledávač Elasticsearch, nástroj pro orchestraci kontejnerů Docker, verzovací systém GIT, webový server nginx a především jsem si prohloubil znalosti operačního systému Linux v distribucích Ubuntu a Debian.

V rámci mé pozice a trochu nad její rámec jsem měl navrhnout, naimplementovat a vyzkoušet IoT bránu postavenou na mnou vybraném hardwaru a softwaru a vyzkoušet její praktickou aplikaci pro řešení vzorového úkolu.

2 Seznam zadaných úkolů

2.1 Výběr vhodné referenční HW platformy pro bránu

Časový odhad: 80 hodin

Cílem je vhodně vybrat hardware pro IoT bránu, na které se bude vyvíjet a testovat celé řešení a seznámit se s ní. K hardwaru by měla být k dispozici dokumentace a schéma zapojení.

Součastí úkolu je také výběr z dostupných bezdrátových modulů pro komunikaci s koncovými zařízeními (senzory) a seznámení se s nimi.

2.2 Výběr operačního systému pro bránu

Časový odhad: 20 hodin

Pro zvolenou hardwarovou platformu vybrat vhodný operační systém, založený na Linuxu, následně navrhnout a vyzkoušet konfigurace pro stabilní nepřetržitý provoz.

Součástí úkolu je instalace a konfigurace virtualizační platformy Docker na zvoleném operačním systému a návrh řešení, založeného na kontejnerové virtualizaci.

2.3 Tvorba aplikačního rozhraní

Časový odhad: 100 hodin

Implementace jednoduchého backend rozhraní, postaveného nad protokolem HTTP a databázovým systémem dle vlastního výběru zajišťující řízení jednotlivých bran, jejich registraci, správu a aktualizace.

2.4 Implementace a ladění obslužné aplikace

Časový odhad: 120 hodin

Tvorba aplikace, zajišťující provedení konfiguračních změn, provedených v aplikačním rozhraní, na samotných branách. Aplikace bude komunikovat pomocí definovaného aplikačního rozhraní z úkolu 2.3.

2.5 Zabezpečení infrastruktury

Časový odhad: 50 hodin

Výstupem úkolu bude celková revize zabezpečení celé infrastruktury bran podle běžně používaných praktik a případně návrhy na vylepšení.

2.6 Praktická aplikace

Časový odhad: 40 hodin

Cílem úkolu je vymyslet, navrhnout a sestavit běžný testovací scénář pro infrastrukturu, navrženou v předchozích zádáních. Data z čidel budou sbírána do databáze a vykreslována do grafů.

3 Zvolený postup řešení zadaných úkolů

3.1 Výběr vhodné referenční HW platformy pro bránu

Protože výběr jednodeskových počítačů je v posledních letech obrovský, musel jsem vybrat nějaký prověřený, ke kterému bude k dispozici dokumentace a dostatek existujících zkušeností ostatních uživatelů. Absolutním vítězem pro toto kritérium je Raspberry Pi 3, který kromě grafického čipu společnosti Broadcom obsahuje dokumentaci ke všem součástím. [1]

3.1.1 Srovnání vybraných jednodeskových počítačů

Z nepřeberné nabídky počítačů vhodných pro vývoj IoT brány jsem vybral 4 zástupce, kteří dle mého názoru stojí za zmínku a v následujících kapitolách provedu jejich porovnání.

Kritérii pro výběr byly následující položky

- Velikost operační paměti
- Počet/výkon procesorových jader
- Podpora výrobce/komunity
- Rozšiřující rozhraní
- Architektura procesoru

Položky jsou seřazeny podle důležitosti při výběru.

3.1.1.1 Raspberry Pi 3

Tento jednodeskový počítač je třetí generací původního modelu. Cílem konstruktérů bylo vytvořit levný počítač pro podporu výuky elektroniky na britských školách. Počítač obsahuje čtyřjádrový procesor Broadcom BCM2837 založený na architektuře ARMv8.

Tento počítač jsem vybral pro IoT bránu jako referenční, protože lze snadno dohledat řešení mnoha problémů a má také velkou podporu komunity, která se za dobu jeho existence vytvořila. Nabízí také dostatek operační paměti (1 GB) pro provoz systému z RAM.

3.1.1.2 BeagleBone Black

Tento jednodeskový počítač jsem chtěl použít původně, oproti Raspberry Pi 3 nabízí sice poloviční paměť RAM, velkou výhodou je na desce umístěná paměť eMMC o velikosti 4 GB. Na desce je osazen CPU Texas Instruments AM335x taktovaný na frekvenci 1 GHz, doplněn je 512 MB operační paměti.

Nevýhodou je vyšší pořizovací cena, v době vypracovávání této práce tento počítač také oficiálně neprodával žádný prodejce v ČR.

3.1.1.3 OrangePi Plus

OrangePi Plus má být levnější náhrada Raspberry Pi 3 s větší výbavou. Mezi tu patří např. SATA řadič pro připojení pevných disků, integrovaná eMMC flash paměť o velikosti 8 GB, IR přijímač pro dálkové ovládání a adaptér Ethernet standardu 1000Base-T. Problémem je nepříliš kvalitní výstupní kontrola výrobce a rozdíly i v rámci jedné výrobní série, na které si stěžují uživatelé napříč diskuzními fóry. Pro použití jako IoT brána je navíc spousta rozšíření, které tento počítač nabízí naprosto zbytečných.

3.1.1.4 ***Intel Galileo 2***

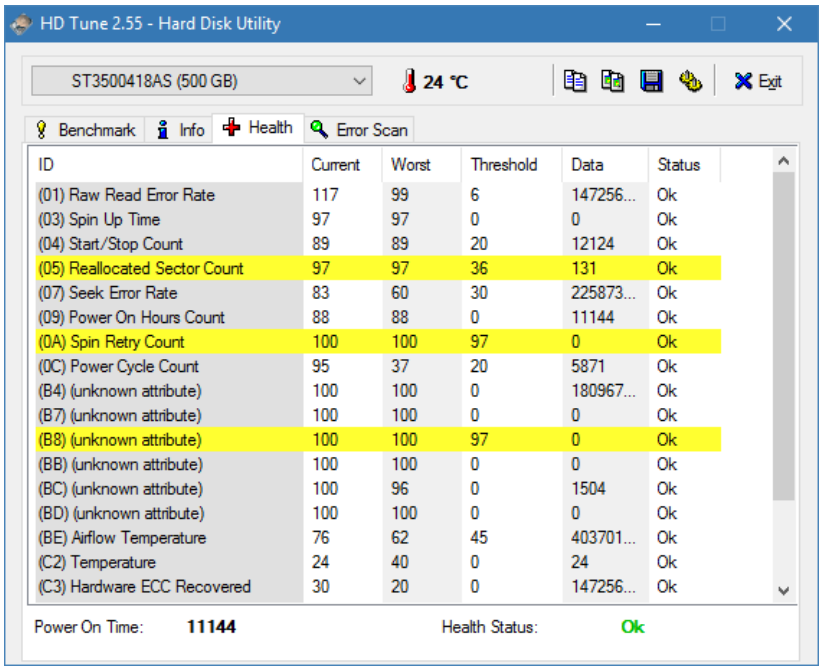
Tato vývojová deska zaujme především architekturou x86, díky které je možné na ní provozovat běžné desktopové/serverové operační systémy. Počítač nabídne veškeré běžné komunikační porty (I2C, SPI, UART). Nabízí však pouze jednojádrový procesor taktovaný na 400 MHz a 256 MB paměti RAM, což v případě použití interpretovaných jazyků s více knihovnami, běžícími v kontejnerech nemusí být dostatečné.

Nevýhodou je také cena, která v konkurenci předchozích počítačů není nejlepší. Nicméně nevylučuji, že další verze IoT brány bude postavena právě na desce společnosti Intel.

3.1.2 Problém nízké životnosti SD karet

Při zkoušení různých distribucí Linuxu docházelo k častým zápisům na SD kartu, která má podle SD Association životnost kolem 10 let. [2] Tato předpokládáná životnost však je však počítaná na zatížení např. v digitálním fotoaparátu, kde jednou za čas dojde k zapsání většího bloku dat. Běžně dostupné SD karty nejsou stavěné na tisíce náhodných přístupů, které běžící operační systém dokáže generovat. Jedná se o různé zápisy do logů, změny konfiguračních souborů, případně nejhorší varianta z pohledu množství zapsaných dat - odkládací prostor SWAP. V případě tohoto typu využití je životnost lepších karet v řádu měsíců, u těch méně kvalitních pak v řádu týdnů.

Řešením pro tento problém jsou průmyslové SD karty, uzpůsobené k provozu ve vyšších teplotách a při větší zátěži. Řadiče těchto karet umí například poskytovat S.M.A.R.T statistiky, podobně jako většina současných pevných disků. Díky tomu je možné včas odhalit, že se blíží konec životnosti a kartu včas vyměnit. Ukázka na obrázku 3.1 pochází z mého PC, je možné vidět, že už mám nějaké realokované sektory.



Obrázek 3.1: *Ukázka výpisu stavových informací S.M.A.R.T pomocí programu HD Tune*

Nevýhodou průmyslových karet je jejich řádově vyšší cena oproti standardním SD kartám a nižší dostupné kapacity. Ačkoli jsem během testování jednu standardní microSD kartu velkým množstvím zápisů zničil, jako lepší varianta se ukázalo nastavit operační systém tak, aby množství zápisů bylo minimální a systém běžel kompletně z operační paměti. Paměťová karta by tak byla využívána pouze k zavádění systému a ukládání změn konfigurace, u kterých by byl požadavek na jejich trvalost - např. aktualizace Linuxového jádra nebo systémových programů. Do produkčního nasazení však potřeba kvalitního úložiště přetrvává, zvláště pokud má brána mít plánovanou životnost okolo 10 let.

Běžnou praxí u standardních SD karet je také sehnat co největší kartu. Vzhledem k tomu, že opotřebení se vztahuje na jednotlivé buňky paměťové karty a řadič rozkládá zápisy na buňky rovnoměrně, čím větší karta, tím pomaleji stoupá její opotřebení. Pro účely vývoje jsem využíval microSD karty o velikosti 32 GB rychlostního standardu UHS-1 od jihokorejského výrobce SAMSUNG.

3.1.3 Výběr bezdrátového rozhraní pro komunikaci se senzory

Ačkoli má být IoT brána maximálně modulární a neměl by být problém se záměnou technologie X za technologii Y, pro účely testování jsem si musel vybrat vhodné bezdrátové moduly a tyto moduly vyzkoušet.

Jelikož cílem tvorby brány bylo pokusit se vyhnout výběru variant využívajících již existující mobilní sítě nebo sítě pro M2M komunikaci - ať už se jedná o různé GSM, LTE modemy, nebo sítě typu LoRa, Sigfox a podobné. Brána má sloužit jako mikrobuňka, obsluhující senzory vzdálené stovky metrů až jednotky kilometrů bez závislosti na síti provozované třetí stranou (s výjimkou sítě Internet). Díky tomu odpadne nutnost platit nemalý paušál za využívání těchto služeb za každé čidlo.

Do úžšího výběru se dostaly moduly uvedené v následujících kapitolách.

3.1.3.1 NORDIC SEMICONDUCTOR nRF24L01

Na tento modul jsem poprvé narazil při svých experimentech s platformou Arduino. Mezi jeho přednosti patří především cena, velmi nízká spotřeba elektrické energie a využívání nelicencovaného pásma 2,4 GHz. Modul obsahuje vysílač i přijímač, komunikovat umí rychlostmi 250 Kb/s, 1 Mb/s a 2 Mb/s, přičemž čím nižší rychlost, tím větší komunikační vzdálenost, která je u nejlevnějšího modulu s anténou na desce plošných spojů několik desítek metrů, u dražších modulů s externí anténou okolo 1 kilometru.

S bránou modul komunikoval pomocí rozhraní SPI, k testu jsem využil knihovnu RF24, vytvářenou uživatelem *maniacbug* a publikovanou na serveru GitHub. Výrobce samotný k modulům nenabízí žádnou volně použitelnou knihovnu a modul samotný neumí žádnou síťovou logiku, proto jsem jej pro použití zavrhnul.

3.1.3.2 Espressif Systems ESP8266

Použít WiFi pro přenos dat mezi senzory se na první pohled zdálo jako výborný nápad. Čínská firma Espressif Systems nabízí procesor s názvem ESP8266, obsahující WiFi ve standardech IEEE 802.11b/g/n společně s pamětí a poměrně výkonným procesorem, taktovaným na 80 MHz. Tento procesor se osazuje na více modulů, mezi nejrozšířenější patří ESP-02.

Modul komunikuje pomocí UART, bohužel využívá 3,3 V logiku, která je nekompatibilní s 5 V logikou využívanou na Raspberry Pi a je potřeba převodník. Modul má také při vysílání poměrně vysoký odběr proudu okolo 300 mA, proto vyžaduje externí napájení.

3.1.3.3 ***IQRF DCTR-72DAT***

Tyto moduly nám do firmy zapůjčil český výrobce a tvůrce technologie IQRF společnost MICRORISC s.r.o. Výrobce kromě hardwaru vyvíjí také vlastní operační systém a implementaci Mesh sítě (smíšená topologie). Nabízí také konektor pro připojení modulu přímo k Raspberry Pi. Moduly pracují v bezlicenčních pásmech 433 nebo 868 MHz (podle typu modulu) s přenosovou rychlostí 20 Kb/s. Maximální velikost jedné zprávy je 64 bajtů, což je pro různá měření dostatečné. Moduly se vyznačují velmi nízkou spotřebou (výrobce udává 15 µA při příjmu).

Jak už jsem zmínil, velkou výhodou je vlastní implementace Mesh sítě, takže odpadá potřeba vyvíjet nebo navrhovat vlastní komunikační řešení, jednotlivé koncové body sítě jsou schopny si mezi sebou předávat data, což v některých situacích může prodloužit dosah, který je mezi dvěma body ve volném prostoru až několik set metrů. Ačkoli je toto řešení na pořízení dražší než všechny předchozí, díky odpadající implementaci vlastního síťového protokolu a nákladům, které by díky vlastnímu návrhu této funkce vznikly, se o až tak drahé řešení nejedná.

Společnost IQRF také založila IQRF Allianci, pod kterou sdružuje výrobce hardwaru i softwaru, a lze tedy do budoucna růst zastoupení této technologie. IQRF Alliance je si vědoma potřeby vzdělávat vývojáře v používání její technologie a množství materiálů, které jsou k dispozici ke studiu je více než dostatečné.

3.2 Výběr operačního systému pro bránu

Důležitým a z pohledu funkčnosti zásadním parametrem IoT brány byl výběr operačního systému. Vybíral jsem z operačních systémů založených na Linuxu, i když pro Raspberry Pi již existuje použitelná alternativa v podobě Microsoft Windows 10 IoT Core. Vzhledem k tomu, že celá současná firemní infrastruktura je postavená na Linuxu a z větší části na svobodném softwaru, zdálo se použití proprietárního produktu, zatím z mého pohledu nedostatečně velkou komunitou okolo, jako neopodstatněné a krokem zpět.

První volba operačního systému padla na systém Raspbian, což je pro Raspberry Pi přizpůsobená odnož Debianu. Tato volba by pasovala k současné infrastruktuře, běžící z větší části na tomto systému. Distribuce nabízí z mého pohledu skvělý balíčkovací systém *dpkg* a jeho nádstavbu *APT*, umožnující snadnou instalaci a aktualizaci nových programů. Instalace platformy Docker taky nebyla problémem, v době vývoje byla verze v repozitářích opožděna asi o jednu nebo dvě vydání.

Systém také plně podporuje DHCP, případně SLAAC pro automatické přidělování adres. Ostatně myšlenka provozovat všechny brány čistě na sítích, podporujících pouze protokol IP verze 6 mi byla poměrně sympatická. Bohužel některé brány vyžadovaly připojení k síti Internet prostředníctvím bezdrátového modemu a v současné době žádný operátor, poskytující služby na území ČR protokol IP verze 6 nepodporuje, na rozdíl od jejich konkurence v USA, kde v roce 2016 podíl dat, přenesených po protokolu IP verze 6 překročil 50 %. [3]

Jedinou nevýhodou Raspbianu je jeho velikost. Systém po instalaci zabírá okolo 1,3 GB, po instalaci dalších balíčků pak jeho velikost ještě naroste.

Distribucí, na kterou jsem narazil při používání Dockeru je Alpine Linux. Tato z mého pohledu minimalistická distribuce obsahuje Linuxové jádro vybavené sadou patchů grsecurity, zajišťující zvýšenou bezpečnost pro kontejnerizovaná řešení a zabraňující některým typům útoků. Je postavená nad odlehčenou knihovnou jazyka C s názvem musl a terminálem BusyBox.

Tvůrci distribuci označují jako nezávislou, nekomerční, všestraně využitelnou distribuci Linuxu pro pokročilé uživatele, kteří ocení jednoduchost, bezpečnost a efektivní využívání zdrojů. [4]

K dispozici je i hotový obraz SD karty pro Raspberry Pi, systém je poněkud atypicky přímo stavěný na zavedení do paměti RAM, persistentní ukládání je řešeno utilitou lbu. Souborový systém je připojený jako smyčka (loop) v paměti nad systémem zavedeným při spuštění z karty, při zavolání příkazu lbu commit jsou do archivu na SD kartu uloženy změněné konfigurační soubory a všechny ostatní definované cesty, které má utilita lbu procházet.

Jelikož jsem se snažil navrhnout hostitelský systém pro kontejnery co nejmenší, Alpine Linux v tomto ohledu docela pomohl. SD karta je využívána minimálně, díky využití archívu s konfiguračními soubory je možné mít celou paměťovou kartu naformátovanou souborovým systémem FAT32 - ostatně pro zaváděci oddíl Raspberry Pi je použití tohoto systému podmínkou. Jako vedlejší výhoda je snadná možnost úpravy konfigurace na kartě z operačního systému Windows, malé usnadnění toto řešení poskytuje i v Linuxu, kdy není potřeba připojovat více oddílů u jedné karty. Kromě archívu s konfigurací se na SD kartu ukládá cache balíčků, protože při zavádění operačního systému dojde k jejich instalaci. Na kartě jsou také uložené obrazy kontejnerů pro Docker, aby při startu nemusely být stahovány z Internetu.

Oproti Raspbianu mi subjektivně přišel bootovací proces Alpine Linuxu rychlejší a přehlednější, Alpine využívá zaváděci systém OpenRC, který je velmi jednoduchý a spouštěcí skripty jsou podobné jako u InitV. Osobně jsem si za poslední dobu zvykl používat spíše Systemd s některými jeho vymoženostmi (časovače, otevíraní socketů), nicméne přechod zpět ke spouštěcím skriptům nebyl velkou překážkou.

Apline Linux využívá vlastní balíčkovací systém APK, v současné době existuje dostatečné množství zrcadel repozitářů a přibývají další, není nutné se bát, že by nebylo odkud stáhnout balíky. Většina běžně používaných programů je v repozitářích k dispozici, některé se od Raspbianu liší pouze jménem. Například balík build-essential, obsahující všechny vývojové nástroje a kompilátory se jmenuje alpine-sdk. Balíčkovací systém také umí aktualizace.

Provozování systému v paměti má i některá úskalí, například při ztrátě napájení dojde i ke ztrátě dat v paměti uložených, nicméně bránu jsem se snažil navrhnout způsobem, kdy veškerá data jsou odesílána dávkově jednou za určitý interval a při jejich ztrátě dojde k minimálním škodám.

Dobré by bylo zmínit, proč jsem se rozhodl postavit celé řešení na kontejnerech v Dockeru a nevytvářet jej přímo nad operačním systémem, ačkoli by to byla s největší pravděpodobností jednodušší varianta. Řešení postavené přímo nad operačním systémem poskytuje vyšší výkon, problém však může nastat při potřebě aktualizace - jakmile si znefunkčním hostitelský systém a nebude možné se k němu vzdáleně připojit (např. překlepem v konfiguraci SSH služby), nezbyde než fyzicky přijít k bráně a opravit chybu ručně. Tato varianta ale nepřichází vždy v úvahu a v některých situacích je finančně náročná - např. při umístění brány na sloupy veřejného osvětlení u frekventované silnice. Tento problém při použití kontejnerů odpadá, protože i samotná SSH služba

může běžet v kontejneru a těchto služeb může běžet i více ve více kontejnerech, což umožňuje si konfiguraci otestovat nanečisto.

S operačním systémem se váže také potřeba tvorby unikátního ID brány, kdy původní záměr byl generovat toto ID a ukládat jej na zvláštní oddíl na SD kartě. Toto řešení se ukázalo jako nevhodné z důvodu zbytečných komplikací při používání této varianty. Jako náhradu jsem využil machine-id, poskytované D-Busem. Machine-id by mělo být globálně unikátní pro každou instalaci této aplikace. D-Bus machine-id je 128 bitové číslo v hexadecimálním tvaru.

3.3 Tvorba aplikačního rozhraní

Aplikační rohraní funguje jako řídící část celého systému. Zajišťuje evidenci všech bran, spravuje seznamy kontejnerů na jednotlivých branách a testuje jejich dostupnost. Zároveň zajišťuje příjem a persistentní uložení měřených hodnot, které projdou přes bránu.

Vzhledem k již existující firemní kultuře jsem se rozhodl využít k implementaci jazyk Ruby, původní myšlenka byla využít i framework Ruby on Rails, vzhledem k jeho rozsáhlosti se mi však nejevil jako optimální řešení. Pro implementaci samotného API jsem si zvolil webový framework Sinatra - ten by se dal označit spíše jako doménově specifický jazyk pro interakci nad HTTP.

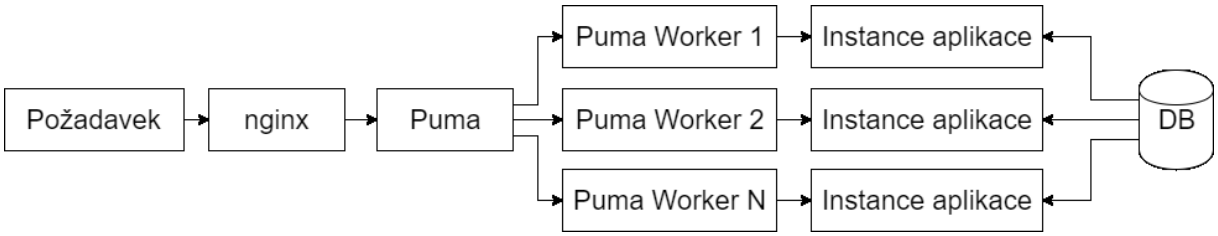
Jednoduchá aplikace v tomto frameworku může vypadat například následovně

```
require 'sinatra'

get '/' do
  'Tento text se vypíše při otevření adresy http://localhost:4567'
end
```

Pro účely produkčního nasazení však z důvodu bezpečnosti není úplně vhodné tento framework spouštět přímo, ale skrz reverzní proxy, ve firmě mnoho využívaný webserver nginx. Při testování se také ukázalo, že při naslouchání aplikace v Sinatře na UNIX domain socketu je zpracování požadavku rychlejší, než při naslouchání na TCP socketu.

Protože framework Sinatra nevyužívá multi-threading a při zpracování několika desítek až stovek požadavků za sekundu nemusí výkon jedné instance aplikace dostatečný, využívá se místo interního webového serveru frameworku Sinatra webový server Puma, který si spustí několik instanci aplikace a požadavky mezi nimi distribuje rovnoměrně. Tím se efektivně využije vysoký počet jader moderních serverových procesorů, princip fungování je popsán na obrázku 3.2. Nevýhodou tohoto návrhu je množství vytvořených a udržovaných spojení na databázový server. Tento problém řeší většina adaptérů pro běžně dostupné databáze, bohužel ne mnou vybraný. Naštěstí existuje knihovna connection_pool, která je natolik univerzální, že ji bylo možné s databázovým systémem Crate využít.



Obrázek 3.2: *Tok zpracování požadavku*

Při výběru databázového serveru jsem si buď mohl vybrat standardní variantu v podobě MySQL nebo PostgreSQL, nebo si vyzkoušet nějaký nový přístup. Po krátkém hledání použitelných variant jsem narazil na poměrně mladý projekt s názvem Crate, který kromě ukládání standardních relačních dat umí efektivně ukládat i časové řady, což při sběru masivního množství dat z různých senzorů může být užitečné. Na časově řady jsem původně chtěl využít InfluxDB, ale Crate to zvládl obdobně. Pokud se toto řešení neosvědčí, lze s menším úsilím přejít na kombinaci MySQL pro relační data a InfluxDB pro ukládání časových řad.

Oproti MySQL však Crate nabízí jednoduchou možnost horizontálního škálování, pokud jedna instance na jednom serveru nebude výkonově dostačovat, spustí se dalších několik serverů a Crate, po nezbytné replikaci dat na nově připojené uzly, automaticky rozloží zátěž na tyto stroje. Pokud se další uzly připojí i z jiných datacenter, je z určité míry zajištěna odolnost proti výpadku. Zátěž se může na fyzických serverech rozkládat například pomocí aplikace HAProxy, v cloudových řešeních se přímo nabízí Load Balancer v cloudu společnosti Amazon.

Pro ukládání dat, posbíraných z jednotlivých čidel jsem využil separátní API, také postavené na Sinatře a databázovém backendu Crate. Jelikož tato data mají trvale stejný formát a jediné, co se mění jsou hodnoty a časová složka, je možné ukládání zefektivnit tak, že v databázi držím data za posledních 7 dní, pokud chci data starší, databázový backend drží uloženou aproximační funkci, která zajistí, že dostanu přibližný výsledek skutečného stavu, aniž by databáze musela data skutečně držet.

API funguje na HTTP, formátem posílaných zpráv je JSON. Autentizace koncové brány probíhá na základě identity, poskytnuté v klientském certifikátu, podepsaném vlastní certifikační autoritou.

3.4 Implementace a ladění obslužné aplikace

Při implementaci samotné obslužné aplikace, běžíci na všech branách, jsem musel vyřešit několik problémů. Prvním z nich byla volba způsobu získávání nových nastavení. V zásadě jsem rozhodoval mezi ověřováním pomocí naplánované úlohy v cronu, nebo procesem, který poběží trvale a bude kontinuálně ověřovat změnu nastavení vůči serveru. Vzhledem k tomu, že jsem se rozhodl řízení kontejnerů v Dockeru implementovat jako kontejner, zvolil jsem první variantu, s tím, že v tabulce naplánovaných úloh hostitelského bude volání skriptu uvnitř řídícího kontejneru. V řídícím kontejneru bude skript v jazyce Ruby, jehož prvním parametrem bude název frekvence, se kterou je volán.

Příklad volání je následující:

```
/usr/bin/docker exec control /opt/cron/cron.rb 15minute
```

Definované frekvence spouštění jsou 5, 15, 30, 60 minut a 1 den. Díky tomu, že je řídící kontejner privilegovaný přistupovat k socketu Dockeru na hostitelském systému, je možné v cronu spouštět i akce pro jiné kontejnery, díky této skutečnosti je spouštění akcí v kontejnerech, zajišťujících napojení na některé externí API velmi jednoduché.

Pro samotné připojení k Docker socketu jsem využil gem (pojmenování knihoven v Ruby) s názvem docker. Tento gem poskytuje třídy pro OOP komunikaci s Docker démonem, což mi umožnilo vyhnout se psaní vlastního adaptéru nebo parsování výstupů z konzole.

Samotné spuštění řídícího kontejneru musí probíhat tak, že je spuštěn jako privilegovaný, aby mohl provádět změny v hostitelském systému a je k němu připojen socket /var/run/docker.sock. [7]

To se provede následujícím příkazem

```
/usr/bin/docker run --privileged \
 -v /var/run/docker.sock:/var/run/docker.sock control
```

Vzhledem k tomu, že většina obrazů, které jsou k dispozici na Docker Hubu, tedy službě poskytující předpřipravené obrazy kontejnerů pro Docker, je určena pro architekturu x86, případně x86_64, bylo nutné vytvořit si vlastní repozitář s předpřipraveným obrazem fungujícího systému pro architekturu ARM.

Jako základní obraz jsem se opět rozhodl využít Alpine Linuxu, s tím, že jsem využil *Minimal root filesystem*, poskytovaný tvůrci Alpine Linuxu přesně pro účely kontejnerové virtualizace. Tento obraz je poskytován jako archiv a obsahuje naprosto základní nástroje, jako správce balíků APK, BusyBox a nejdůležitejší knihovny - libc (v tomto případě odlehčená verze musl), libssl pro zajištění možnosti stahovat další balíky přes SSL spojení, libcrypto pro základní kryptografické funkce a libz pro zajištění komprimačních a dekomprimačních funkcí.

Celý základní obraz se dá zapsat jako jedinný Dockerfile, což je soubor, obsahující "recept" k tvorbě obrazu. Tento soubor má v případě základního obrazu naprosto minimalistický zápis, který tvoří pouze tři řádky.

```
FROM scratch
ADD minirootfs.tar.gz /
CMD ["/bin/sh"]
```

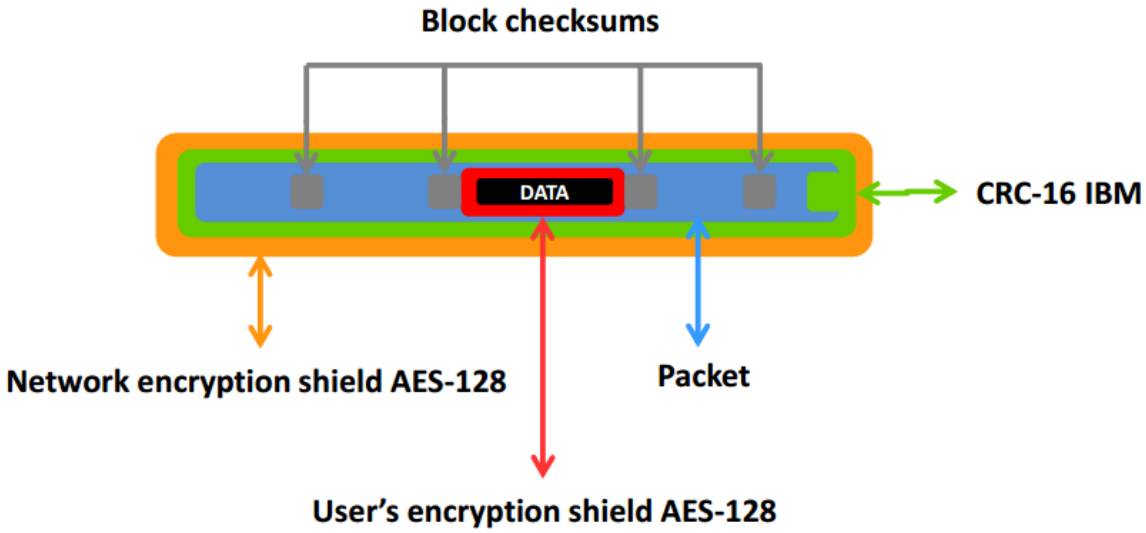
Příkaz *FROM scratch* říká build systému Dockeru, který obraz má využít pro tvorbu nového obrazu. Scratch je v tomto případě speciální případ, protože se jedná o předka všech dalších kontejnerů v Dockeru. Příkaz *ADD minirootfs.tar.gz /* zajistí přidání a překvapivě i rozbalení archivu se základním systémem do kořenového adresáře. Soubor minirootfs.tar.gz byl stažen se stránek Alpine Linuxu. To že byl kontejner automaticky rozbalen mne poněkud překvapilo, nicméně po prostudování dokumentace jsem došel k závěru, že je to očekávatelné chování. Pokud bych chtěl tento archív do nově tvořeného obrazu nakopírovat, je vhodnější použít příkaz *COPY*. Příkaz *CMD* pak řeší, co se má stát při spuštění kontejneru. Vzhledem k tomu, že základní obraz neobsahuje žádné programy navíc, jeví se jako nejlepší volba nechat spustit shell.

Aplikační rozhraní poskytuje obslužné aplikaci seznam kontejnerů, které mají na dané bráně běžet, ve formátu JSON, vzhledem k návrhu systému tak, aby zapisoval co nejméně do úložiště se obrazy kontejnerů stahují při každém restartu brány. Naštěstí při velikosti základního obrazu kolem 5 megabajtů nepředstavuje toto stahování nijak náročný problém (s ohledem na rychlosti, dosahované v současných sítích).

3.5 Zabezpečení infrastruktury

Zabezpečení infrastruktury jsem řešil na třech úrovních, první je zabezpečení přenosu dat od jednotlivých čidel, jejich jednoznačná identifikace a nemožnost změny přenášené informace po cestě mezi čidlem a bránou. Druhou úrovní zabezpečení je komunikace mezi bránami a infrastrukturou, zajištující ukládání, zpracování a zobrazování dat - v tomto případě jsem zvolil cloud, postavený na infrastruktuře vCenter společnosti VMWare. Třetí úrovní je zabezpečení samotných uložených dat proti selhání média, případně selhání fyzického serveru, na kterém běží vCenter.

Zabezpečení přenosu mezi senzorem a bránou vyřešil výrobce IQRF modulů v poslední aktualizaci jeho firmwaru. Řeší jak šifrování sítě, tak šifrování přenesených dat. K tomu využívá šifru AES-128 v režimu CBC (Code block cipher). Z obrázku 3.3 je patrné že nejdříve dojde k zašifrování dat pomocí uživatelského klíče, ten může mít až 16 znaků v ASCII, poté dojde k výpočtu cyklického redundantního součtu (CRC-16) o délce 16 bitů ze zašifrované zprávy a údajů z hlavičky rámce, následně dojde k finálnímu zašifrování celého rámce síťovým klíčem, opět maximální délka klíče je 16 ASCII znaků. V celém procesu dále figuruje ještě přístupový klíč (Access key), zajišťující ověření stanice při párování do mesh sítě.



Obrázek 3.3: *Schéma šifrování přenášeného rámce [5]*

Pro zabezpečení přenosu dat od brány k serverům jsem použil protokol HTTPS a ověřuji jak identitu serveru, tak identitu každé jednotlivé brány. Pro ověřování identity serveru jsem využil certifikátů, podepsaných certifikační autoritou Let's Encrypt, pokud by však v budoucnu došlo k zániku této autority, nebude problém přejít na jinou, buď komerční, nebo vlastní. V případě komerční autority by stačilo aktualizovat v hostitelském systému balíček *ca-certificates*, obsahující certifikáty důvěryhodných autorit - autoři Alpine Linux je přebírají z Mozilly, v případě vlastní autority by bylo potřeba vytvořit balíček, podobný předchozímu, obsahující certifikát mnou generované autority, což by bylo složitější, ale nikoli neřešitelné.

Ověření identity klienta probíhá přihlašováním klientským SSL certifikátem, podepsaným mnou vytvořenou certifikační autoritou. Při návrhu jsem se inspiroval protokolem, využíváným při automatickém vydávání certifikátů autoritou Let's Encrypt. Brána si vygeneruje privátní klíč pomocí OpenSSL, pro toto využití jsem se rozhodl generovat klíče, založené na eliptických křivkách, protože při kratším klíči poskytují vyšší bezpečnost, jejich generování je méně náročné na výkon CPU a velikost RAM. [6]

Problém nastal v okamžiku, kdy jsem chtěl využít adaptér pro OpenSSL v jazyce Ruby a narazil jsem na nedokončenou implementaci podpory klíčů, založených na eliptických křivkách. Z tohoto důvodu jsem byl nucem přejít na šifrování asymetrickou šifru RSA, pro vyvážení bezpečnosti a náročnosti na výkon jsem zvolil délku klíče 2048 bitů. Do nástupu kvantových počítačů nebo objevu zásadní bezpečnostní trhliny v návrhu algoritmu RSA bude tato délka dostatečná.

Ke generování klíčů využívám knihovnu OpenSSL, respektive její adaptér pro Ruby. Jelikož jsem v této implementaci narazil na problém s generováním požadavku k podpisu veřejného klíče, rozhodl jsem se místo ECC využít RSA. Kód pro generování požadavku k podpisu vypadá následovně:

```
require 'openssl'

id_file = '/var/lib/dbus/machine-id'

key = OpenSSL::PKey::RSA.new 2048

csr_options = {
    :country      => 'CZ',
    :state        => 'Moravskoslezsky',
    :city         => 'Ostrava',
    :department   => '',
    :organization => 'Railsformers',
    :common_name  => File.read(id_file),
    :email        => ''
}

request = OpenSSL::X509::Request.new

request.version = 0
request.subject = OpenSSL::X509::Name.new([
        ['C',             csr_options[:country]]
        ['ST',             csr_options[:state]]
        ['L',              csr_options[:city]],
        ['O',              csr_options[:organization]],
        ['OU',             csr_options[:department]],
        ['CN',             csr_options[:common_name]],
        ['emailAddress',  csr_options[:email]]
    ])

request.public_key = key.public_key

request.sign(key, OpenSSL::Digest::SHA256.new
```

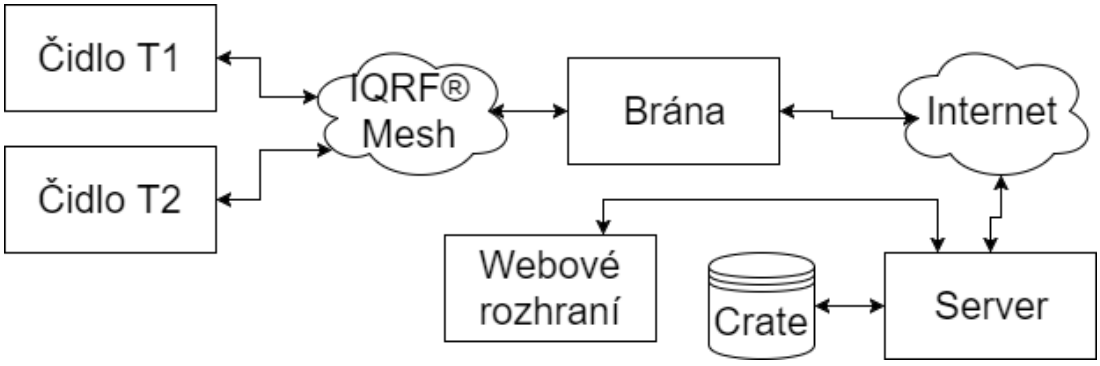
Tento kód provede vygenerování nového privátního klíče RSA o délce 2048 bitů, následně pomocí parametrů, uvedených v proměnné csr_options vytvoří požadavek na podpis, identifikátorem certifikátu následně bude machine ID, načtené ze souboru.

Přenos CSR následně probíhá pomocí HTTP požadavku na server, bohužel v současné verzi systém neumožňuje zjištění, zda klient, požadující podepsání certifikátu je skutečně ten, za kterého se vydává. To je zatím ošetřeno manuálním schvalováním každého nového zařízení. Při požadavku na plně automatické nasazování bude implementován ověřovací algoritmus, založený na algoritmu Diffie-Hellman pro bezpečnou výměnu klíčů.

Zabezpečení na úrovni serveru je řešeno z více stran. Jako zabezpečení proti selhání média jsou pevné disky zapojeny replikované do svazku RAID-5, selhání fyzického serveru pak řeší samotný vCenter, kdy úložiště je fyzicky jinde než uzel, na kterém běží operační systém, v případě selhání tohoto uzlu je operační systém znovu spuštěn na jiném fyzickém stroji. Díky využití replikovaného databázového systému Crate je možné provozovat několik instancí serverové aplikace na několika serverech. Pro účely testování jsem využil dva virtuální stroje, zátěž je rozkládána pomocí systému DNS kdy jméno serveru má více A záznamů a klient vybírá náhodně pomocí algoritmu round-robin. Tento způsob nefunguje v některých především starších prohlížečích, které vyberou pouze jeden A záznam. To poskytuje určitou míru tolerance k selhání jednoho ze serverů. Servery samotné jsou zálohovány 2x denně přírůstkovou zálohou, jednou týdně je provedena kompletní záloha obsahu serverů. Bezpečnost databáze proti selhání hardware je řešena replikací dat na více uzlů, proti selhání lidského faktoru (smazání dat, ať už úmyslné nebo omylem) je řešeno exportem komprimované zálohy databáze každé dvě hodiny - zde se projevuje výhoda distribuovaného databázového systému, protože díky vyvažování zátěže nemá samotné zálohování prakticky žádný či spíše minimální vliv na výkon aplikace.

3.6 Praktická aplikace

Praktická aplikace se skládá z několika prvků, popsaných v obrázku 3.4. Vzhledem k tomu, že mnou použité IQRF moduly mají vestavěný teplotní senzor, bylo nejjednodušší možné využití měření teploty v několika místnostech. Měl jsem k dispozici tři IQRF moduly - dva jsem využil k měření teploty, jeden jsem využil jako komunikační rozhraní k bráně.



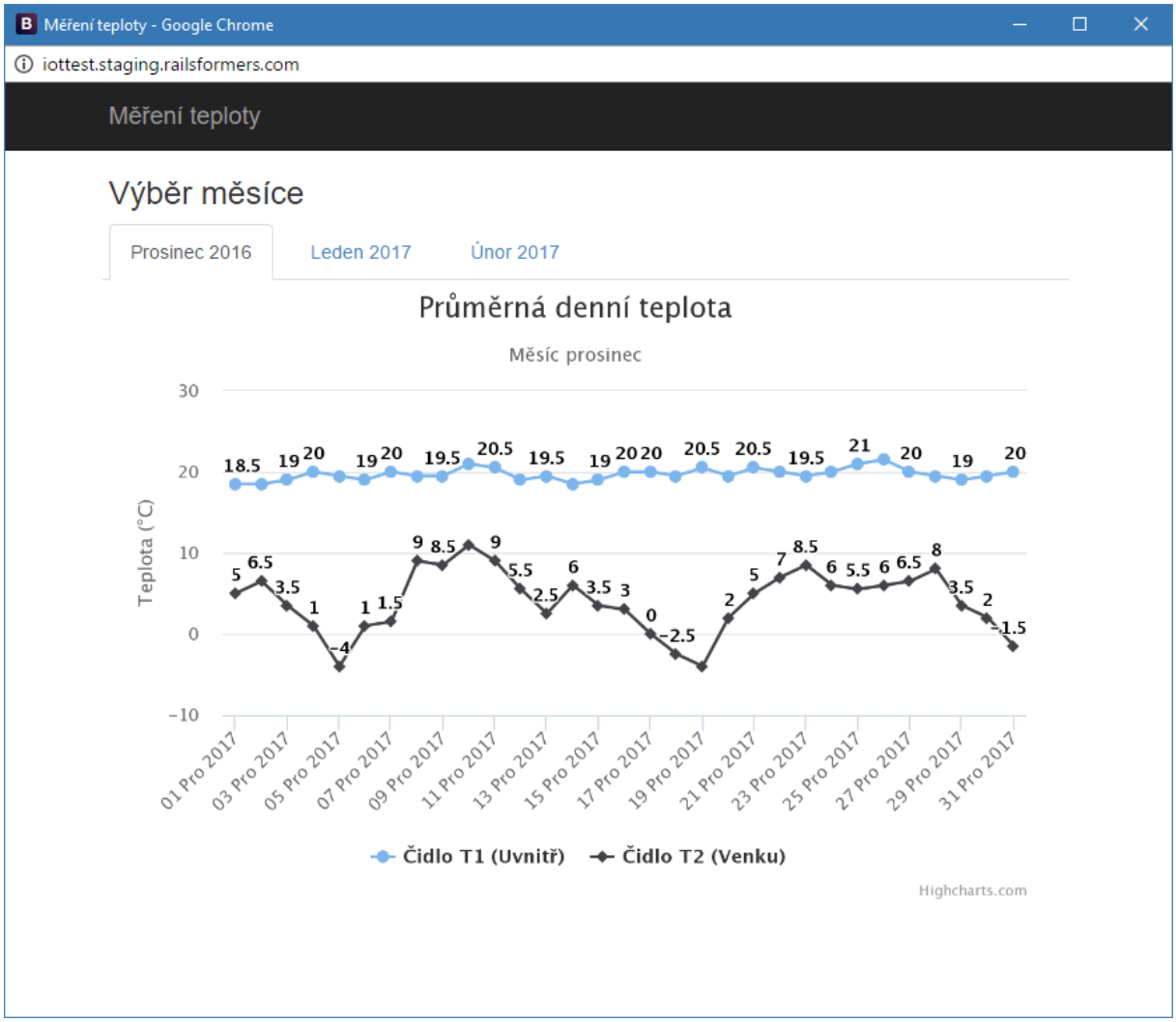
Obrázek 3.4: *Schéma testovací aplikace*

V bráně kromě řídícího kontejneru běží ještě kontejner, obsluhující SPI rozhraní, komunikující s IQRF modulem. Tento program dodala třetí strana a je dodáván jako binární soubor kompatibilní s architekturou ARMHF. Po jeho spuštění dojde k odeslání příkazu všem v síti připojeným IQRF čidlům, následně dojde asi k čekání na odpovědi o délce trvání 30 sekund a vrácení výsledků. Výstup programu je pole ve formátu JSON, které vypadá následovně

```
/ # /usr/bin/gettemperatures -d /dev/spidev0.0
```

```
{"data": [{"station_id" : "1", "temperature": "18.5"}, {"station_id"
: "2", "temperature": "17.5"}]}
```

Tento výstup následně zpracovávám ve skriptu v Ruby a pomocí třídy Net::HTTP odesílám na server. K připojení a jednoznačné identifikaci brány se zde využívá klientský SSL certifikát. Odeslaná data jsou následně uložena na serveru do databázového systému Crate a využívají se v grafickém uživatelském rozhraní, realizovaném jako webová stránka, napsaná ve frameworku Boostrap. Pro grafické znázornění uložených hodnot využívám knihovnu Highcharts. Výsledek je zobrazen v obrázku 3.5. Pro vytvoření API k přístupu k datům, uloženým v databázi byl opět vybrán framework Sinatra, kdy pomocí požadavku GET s URL parametry dateFrom a dateTo je navrácen JSON objekt, obsahující jednotlivé záznamy. Pro připojení k databázi využívám adaptéru crate_ruby, pro větší aplikace je možné využít frameworku Ruby on Rails a adaptéru do Active Record, což je označení databázové vrstvy Ruby on Rails.



Obrázek 3.5: *Náhled webového rozhraní*

Celá stránka je vykreslená na základě jednoho dotazu na databázový server, další data (například při změně měsíce) jsou stahována na pozadí a jejich vykreslení probíhá za běhu. Webové rozhraní je velmi prosté, cílem bylo ukázat, že celý systém funguje, nikoli komplexní aplikaci pro práci s naměřenými daty.

Kromě webového rozhraní pro zobrazování výsledků jsem ještě považoval za praktické vytvořit jednoduchou administraci jednotlivých čidel a bran. Tu jsem opět postavil na frameworku Bootstrap a obsahuje základní funkcionalitu, nutnou pro funkčnost systému, tedy seznam nových bran ke schválení, poslední data z čidel, správa obrazů a běžících kontejnerů jednotlivých bran. Toto rozhraní však obsahuje pouze nejnutnější funkcionalitu, v budoucnosti, společně s požadavky na další funkcionalitu dojde k přepsání aplikace do frameworku Ruby on Rails. Zároveň pravděpodobně dojde k vytvoření pokročilejšího rozhraní pro práci s naměřenými hodnotami, rozhraní pro definování událostí, založených na naměřených hodnotách (např. zavolání externího API, zaslání SMS) přímo v obslužné aplikaci pro správu jednotlivých bran, aby byla poskytnuta komplexní funkcionalita bez nutnosti vyvíjet speciální nádstavby, ačkoli tato možnost bude stále k dispozici.

4 Závěr

4.1 Teoretické a praktické dovednosti a znalosti uplatněné v průběhu praxe

Při absolvování praxe ve společnosti Railsformers s.r.o. jsem využil a zároveň si rozšířil znalosti z následujících předmětů

- **Databázové a informační systémy**
  Znalosti z tohoto předmětu jsem uplatnil při výběru databázových systémů pro ukládání dat poskytovaných senzory a při využívání knihoven pro objektově-relační mapování.

- **Vývoj informačních systémů**
  Znalosti z tohoto předmětu jsem uplatnil především při návrhu, jak má celá aplikace vypadat. Ukázalo se, že znalost různých návrhových vzorů umožnuje psát čistší, přehlednější a efektivnější kód.

- **Vývoj internetových aplikací**
  Znalosti z tohoto předmětu jsem využil především při tvorbě webových stránek, založených na frameworku Bootstrap a při implementaci logiky, psané v JavaScriptu.

- **Správa operačních systémů**
  Tento předmět mi poskytl přehled o fungování operačních systémů, založených na Linuxu a možnostech, které poskytuje počítač Raspberry Pi.

- **Skriptovací a programovací jazyky a jejich aplikace**
  Některé znalosti, poskytnuté tímto předmětem pro jazyk Python jsou přenosné i do jazyka Ruby.

- **Architektury počítačů a paralelních systémů**
  V rámci tohoto předmětu jsem získal přehled o jednotlivých architekturách procesorů a prakticky si vyzkoušel komunikaci pomocí některých nízkoúrovňových protokolů (např. I2C).

- **Počítačová bezpečnost**
  Tento předmět mi poskytl přehled v oblasti bezpečnosti sítí a kryptografie.

4.2 Znalosti a dovednosti scházející v průběhu praxe

Jedním z pilířů celé této bakalářské práce je virtualizační platforma Docker. Ačkoli se jedná o poměrně novou věc, myslím, že by nebylo na škodu kdybych se o ní něco dozvěděl i při studiu. Další znalostí, kterou jsem se musel doučit sám byl verzovací systém GIT, ačkoli na naučení se této znalosti apeloval cvičící v předmětu Programovací jazyky 1.

Musím však uznat, že teoretický základ, poskytnutý na VŠB-TUO je dostatečný, vzhledem k množství nově vzniklých a vznikajících technologií není ve schopnostech žádné univerzity na světě pokrýt veškeré tyto technologie.

4.3 Dosažené výsledky v průběhu praxe a její celkové zhodnocení

Kdybych měl stručně shrnout dosažené výsledky mé praxe, řekl bych, že jsem vcelku úspěšně navrhnul architekturu celého systému a určil směr, kterým by se systém měl ubírat. V současné době určitě není systém ve stavu, kdy by jej bylo možné prodávat, jde spíše o ukázku toho, že to může fungovat. Určitě by bylo dobré napsat aplikační rozhraní ve frameworku Ruby on Rails, čímž by se zajistila lepší rozšiřitelnost, protože Sinatra svádí k psaní špatného kódu. Do produkční verze bych určitě nepoužíval Raspberry Pi, ale šáhnul po osvědčeném řešené některého z mnoha výrobců procesorů a velmi vhodné by bylo postavit celou architekturu na platforme x86, protože kompilace aplikací v QEMU případně přímo na Raspberry je pro účely vytváření nových aplikací do kontejnerů nepříjemná a zdlouhavá. Určitě by ještě bylo vhodné zapracovat na síťovém systému, protože v současné podobě brána spoléhá na získání IP adresy z DHCP serveru. Vhodné by také bylo sesíťovat brány do jedné podsítě pomocí virtuální privátní sítě. Jako databázi pro ukládání časových řad bych dnes zcela určitě využil InfluxDB, případně nějaký podobný software, protože tato vrstva v databázovém systému Crate je spíše vrstva navíc, než použitelná funkčnost.

Internet věcí bude v budoucnu stále častěji skloňovaným tématem, už i v současnosti je možné nalézt mnoho schopných projektů, ale i hotových produktů. Jsem si jistý, že společnost Railsformers bude v tomto směru leaderem minimálně na regionální úrovni.

Použitá literatura

[1] Raspberry Pi Hardware. *Raspberry Pi Documentation* [online]. Cambridge [cit. 2017-03-27]. Dostupné z: https://www.raspberrypi.org/documentation/hardware/raspberrypi/README.md

[2] FAQs. *SD Association* [online]. San Ramon [cit. 2017-03-27]. Dostupné z: https://www.sdcard.org/consumers/faq/#servicelife

[3] MAJOR MOBILE US NETWORKS PASS 50% IPV6 THRESHOLD. *World IPv6 Launch* [online]. Reston, Virginia, USA, 2016 [cit. 2017-03-28]. Dostupné z: http://www.worldipv6launch.org/major-mobile-us-networks-pass-50-ipv6-threshold/

[4] About. *Alpine Linux* [online]. 2016 [cit. 2017-03-28]. Dostupné z: https://alpinelinux.org/about/

[5] User Encryption. In: *IQRF* [online]. Jičín, 2017 [cit. 2017-04-09]. Dostupné z: http://www.iqrfalliance.org/data_files/news/2017-03-30-iqrf---os-4.0-webinar.pdf

[6] Elliptic Curve cryptography. *GlobalSign Blog* [online]. Portsmouth, USA, 2015 [cit. 2017-04-09]. Dostupné z: https://www.globalsign.com/en/blog/elliptic-curve-cryptography/

[7] TURNBULL, James. *The Docker Book*. V1.0.7 (8f1618c). New York, 2014. ISBN 9780988820203.