Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra číslicového návrhu

Diplomová práce

Nadřazený systém pro správu garáže

Bc. Ondřej Červenka

Vedoucí práce: Ing. Martin Daňhel

14. února 2018

Poděkování

Děkuji panu Ing. Martinu Daňhelovi za čas, který mi věnoval a zejména za cenné rady a odborné vedení mé diplomové práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 14. února 2018 .....

Odkaz na tuto práci

Červenka, Ondřej. *Nadřazený systém pro správu garáže.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

V několika větách shrňte obsah a přínos této práce v češtině. Po přečtení abstraktu by se čtenář měl mít čtenář dost informací pro rozhodnutí, zda chce Vaši práci číst.

Klíčová slova Nahraďte seznamem klíčových slov v češtině oddělených čárkou.

Abstract

Sem doplňte ekvivalent abstraktu Vaší práce v angličtině.

Keywords Nahraďte seznamem klíčových slov v angličtině oddělených čárkou.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Cílem této práce je vytvořit nadřazený systém pro monitorování garážového komplexu. Výsledná aplikace bude komunikovat pomocí WiFi či Ethernetu s podřízenými systémy (zasílajícími údaje z čidel v garážích). Na základě získaných dat pak bude udržován stav jednotlivých garáží a vytvářena historii událostí.

Systém bude poskytovat webového rozhraní pro administraci. V tom bude možné přidávat a odebírat podřízené systémy, zobrazovat jejich stav a zaznamenané události.

Vzhledem k povaze zadání je nutné systém navrhnout s ohledem na zabezpečení přenášených informací před odposloucháváním či manipulací. Též je nutné autorizovat uživatele přistupující do webového rozhraní.

Dalším důležitým požadavkem je snadná rozšiřitelnost o nové funkce. Systém by mělo být možné v budoucnu doplnit o možnost správy rozdílných podřízených systémů (například subsystémy pro sledování skladových zásob) či integraci s mobilní aplikací. Bude tedy potřeba navrhnout vhodné API pro předávání informací mezi systémem a jeho klienty.

V této práci se chci zaměřit na tvorbu aplikace na jedné konkrétní hardwarové platformě, jako například Raspberry Pi. Aplikace spolu s touto platformou by pak měla tvořit kompletní zařízení, které bude možné po základní konfiguraci (připojení do WiFi sítě, nastavení hesla) hned nasadit.

Výsledné řešení by však mělo být dostatečne nezávislé na zvolené platformě. Tudíž by neměl být problém spustit systém například na osobním počítači či virtuálním serveru.

V analytické části (1) práce tedy přiblížím proces výběru vhodné platformy, komunikačního protokolu a dalších prvků systému. Také stručně popíšu podřízený systém (garážové čidlo), se kterým budu dále pracovat. Další části mapují návrh (2) systému na základě této analýzy, jeho implemetaci (3), nasazení (4) na zvoleném hardwaru a testování (5).

Kapitola 1

Analýza

1.1 Struktura systému

Struktura celého systému je naznačena na obrázku 1.1. Podřízené systémy komunikují s nadřazeným na základě událostí. Nadřazený systém tyto události zpracovává a upravuje podle nich stav garáží v evidenci.

Zaznamenané události jsou také uchovávány v historii událostí, spolu s dalšími metadaty jako čas přijetí nebo původce.

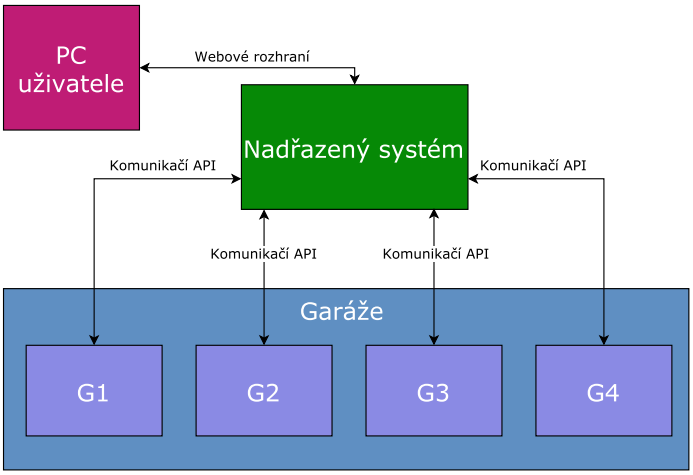
Komunikace mezi podřízeným a nadřazeným systémem je postavena na modelu *client/server*. Nadřazený systém provozuje server zvoleného protokolu (viz sekce 1.2), ke kterému se podřízené systémy připojují. Komunikaci tedy vždy iniciuje podřízený systém. S možností zasílání nevyžádaných zpráv podřízeným systémům v této práci nepočítám, mohl by to však být námět pro další rozšíření.

Další, kdo přistupuje do systému, je uživatel. Přes webové rozhraní může sledovat stav garáží a historii událostí. Také zde může spravovat klíče, které slouží pro přístup ke komunikačnímu API systému. Přístup do webového rozhraní je zabezpečen heslem.

1.1.1 Podřízený systém

Podřízený systém je zařízení umístěné v každé garáži, které sleduje stav okolí pomocí těchto senzorových vstupů:

- teplota,
- světelá intenzita (fotobuňka),
- detekce kouře,
- detekce pohybu,
- stav dveří.



Obrázek 1.1: Základní struktura systému

V případě překročení mezních hodnot se zařízení okamžitě hlásí nadřazenému systému. Kromě toho také v pravidelných intervalech odesílá kontrolní hlášení.

Vyhodnocení události je provedeno nadřazeným systémem. Podřízený systém tedy hlásí každou událost (například otevření dveří), aniž by nějak zkoumal její závažnost.

Základní požadavek na podřízený systém je schopnost komunikace přes Ethernet či WiFi pomocí protokolu zvoleného v sekci 1.2. Kromě toho může být hardware prakticky libovolný.

1.2 Výběr komunikačního protokolu

Nejdřív je nutné určit způsob komunikace, který bude systém používat. Díky tomu se budu při vybíraní platformy moci ujistit, že jsou dostupné vhodné knihovny a další software.

Nadřazený systém bude se svými klienty (monitorovací zařízení v jednotlivých garážích) komunikovat přes WiFi nebo Ethernet. Základem komunikace bude TCP/IP protokol, je však potřeba zvolit vhodný protokol z aplikační vrstvy OSI modelu, který na něm bude stavět.

Při výběru protokolu jsem vycházel z předpokaldu, že systém bude provozován v uzavřené síti a bez přístupu k internetu.

1.2.1 Vlastní protokol

Jedna z možností je implementovat vlastní protokol pomocí TCP/IP socketů. Toto řešení se mi však nezdá příliš vhodné, neboť nepřináší žádné významné výhody, naopak se s ním pojí řada komplikací.

Pro vlastní protokol by bylo nutné vytvořit robustní server, který zvládá obsluhu více klientů najednou. Dále by vzhledem k citlivosti přenášených dat bylo nutné implementovat nějakou formu šifrování. Tyto velmi obsáhlé problémy přitom řeší většina dnešních protokolů.

Další nevýhodou je nutnost implementace klientské části protokolu při vytváření nových zařízení spravovaných nadřazeným systémem. To do jisté míry omezuje jeho rozšiřitelnost.

1.2.2 HTTPS

Další možnost je využít ke komunikaci protokol HTTPS. V tomto případě by klienti komunikovali se sytémem pomocí HTTP metod jako například `get` nebo `post`.

Jelikož součástí požadavků na systém je i webové uživatelské rozhraní, bude v každém případě nutné použít webový server pro jeho provoz. Ten by pak bylo možné využít i k poskytnutí API pro komunikaci systému s garážovými čidly.

Vhodný webový server (jako například Apache) zajistí vícevláknovou obsluhu všech klientů. Protokol se také postará o kryptografické zabezpeční přenášených dat, je však nutné získat certifikát k ověření pravosti serveru (viz sekci 1.2.2.1).

Certifikát bude potřeba zajistit i v případě, že komunikace s klienty nebude postavena na tomto protokolu. Je totiž nutné také zabezpečit webové rozhraní, například kvůli ověření identity uživatele. Nutnost pořízení certifikátu tedy nepředstavuje nevýhodu oproti jiným protokolům.

API realizované pomocí tohoto protokolu je poměrně snadno rozšiřitelné. Pro nově implementovanou operaci stačí definovat URL a případně formát přenášených dat.

Výhodou je také snadná implementace na straně klienta, tedy garážového čidla. Knihovny realizující klientskou část protokolu jsou dostupné na většině populárních platforem jako například Arduino (s Ethernet *shieldem*, oficiální knihovna EthernetClient [1]) nebo ESP8266 (knihovna esp8266wifi [2]).

1.2.2.1 Certifikáty pro provoz HTTPS

Pro provoz HTTPS serveru lze použít například certifikáty certifikační autority Let's Encrypt, které jsou poskytovány zdarma. Kromě toho dodává Let's Encrypt také automatizačního klienta Certbot [3] pro snadné nasazení a aktualizaci jejich certifikátů. Bohužel certifikáty jsou vydávány pouze na doménu [4], což komplikuje použití v místní síti.

Jiná možnost je použití *self-signed* certifikátu. Tento certifikát není podepsaný žádnou certifikační autoritou, ale pouze vlastníkem certifikátu. Může tedy sloužit k šifrování komunikace (poskytuje veřejný klíč), ale je zranitelný vůči *man-in-the-middle* útoku [5].

Self-signed certifiát však lze použít k šifrování komunikace na uzavřené lokální síti, za předpokladu, že je server s certifikátem (přesněji s jeho soukromým klíčem) dostatečně zabezpečen [5].

Nevýhodou tohoto řešení je nedůvěra webových klientů (certifikát není podepsán certifikační autoritou a nelze tedy ověřit jeho pravost), což by ovlivnilo přístup k uživatelskému rozhraní a API systému. V případě webového rozhraní by prohlížeč zobrazil varování o neznámém certifikátu. To by však mohl uživatel ignorovat.

Podřízené systémy by při zasílání požadavků museli přeskočit krok ověření totožnosti serveru. Jak toho dosáhnout například v knihovně Requests pro Python je naznačeno v ukázce 1.

```
>>> import requests
>>> r = requests.get('https://test.local/hello', verify=False)
>>> r.status_code
200
```

Ukázka 1: Vytvoření HTTPS požadavku v knihovně Requests, bez verifikace serveru

1.2.2.2 Autentizace klientů na HTTPS

Přístup k API nadřazeného systému by měl být povolen pouze ověřeným klientům. Díky tomu bude možné zabránit například zasílání nepravdivých informací z neznámých zdrojů.

Jednoduchou autentizaci přes HTTPS lze realizovat například pomocí generování API klíčů. Pro každý podřízený systém bude vygenerován klíč, kterým se při zasílání požadavku systém prokáže. Seznam platných klíčů by byl udržován v databázi nadřazeného systému. Klíče by uživatel mohl přidávat nebo odebírat (například v případě odcizení podřízeného systému) pomocí webového rozhraní.

Tyto klíče by také bylo nutné nahrát a uchovávat na podřízených systémech. Detaily tohoto procesu by záležely na platformě těchto systémů. Například u Arduina by šlo klíč nahrát z uživatelova počítače pomocí sériové linky (s USB převodníkem) a udržovat ho v EEPROM.

Také by bylo možné implementovat v nadřazeném systému „registrační mód“, který by bylo možné dočasně povolit ve webovém rozhraní. V tomto módu by systém po přijetí speciálního API požadavku vygeneroval nový klíč. Ten by si uložil do své databáze platných klíčů, a také ho v odpovědi zaslal žádajícímu zařízení. Pokud by mód povolen nebyl, odpovědel by systém chybovým kódem, například 403 – *Forbidden*. Zaslání požadavku z podřízeného systému mohlo být provedeno stisknutím tlačítka.

Tento přístup by byl pravděpodobně uživatelsky příjemnější, přináší však potencionální bezpečnostní rizika. Například pokud by uživatel zapomněl mód

vypnout, systém by byl otevřený k registraci nežádoucích zařízení. Takový problém by se však dal řešit například automatickou deaktivací módu po uplynutí časového limitu.

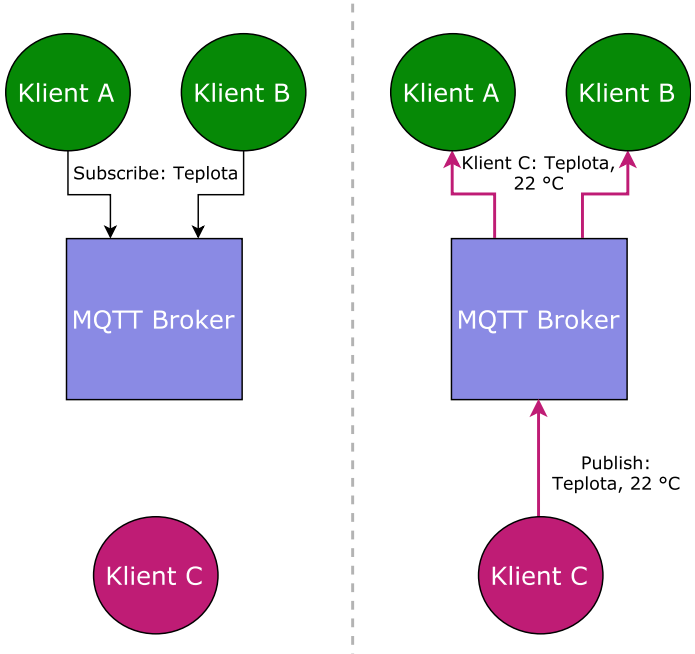
Útočník snažící se získat klíč k API by také mohl periodicky zkoušet registrační požadavek a čekat na aktivaci módu. Obrana proti tomuto útoku by byla složitější, šlo by například filtrovat IP adresy s příliš častými požadavky.

Obecně vycházím z toho, že i v případě registrace nežádoucího zařízení nemůže toto zařízení krátkodobě způsobit výraznější škody – do databáze nadřazeného systému může pouze zasílat nová data, která jsou navíc vázana k jeho identitě (API klíči). Nemůže tedy získávat data od jiných podřízených systému či měnit jejich záznamy. Neautorizované zařízení se také objeví v seznamu registrovaných API klíčů, kde může být snadno odhaleno.

1.2.3 MQTT

MQTT je komunikační protokol založený na modelu *publisher/subscriber*, určený pro použití v prostředí s omezenými zdroji (malý výkon procesoru, omezená paměť atd.) [6].

Komunikace mezi jednotlivými klienty v systému je zprostředkována pomocí centrály, nazývané *broker*. Ta spravuje adresy – *topics* – na kterých mohou klienti publikovat či odebírat zprávy.



Obrázek 1.2: Příklad struktury protokolu MQTT [7]

Na obrázku 1.2 tedy klienti A a B začnou odebírat *topic* „Teplota“. Když pak klient C pak publikuje zprávu na tuto adresu, *broker* se postará o doručení všem odebírajícím klientům.

Adresy je možné hierarchicky strukturovat. Lze tedy tvořit skupiny, například `/senzory/obyvak/teplota` nebo `/senzory/kuchyne/vlhkost`. Zprávy je nutné publikovat na jednoznačnou adresu, při odebírání je však možné použít modifikátory `+` a `#` pro specifikování skupiny adres. Modifikátor `+` odpovídá libovolnému jednomu stupni hierarchie, `#` pak libovolnému počtu libovolných stupňů. Pro odebírání všech senzorů vlhkosti lze tedy použít adresu `/senzory/+/vlhkost`. Všechna data by pak bylo možné odebírat na adrese `/senzory/#`. [7]

V případě této práce by tedy jak nadřazený systém, tak podřízené systémy byly klienty *brokeru*. Podřízené systémy by publikovaly naměřená data, která by nadřazený systém odebíral. Samotný *broker* by pak mohl běžet souběžně s nadřazeným systémem na zvolené platformě (například open-source *broker* Mosquitto je dostupný na řadě platforem, včetně Raspberry Pi [8]).

Protokol podporuje tři možnosti QoS [6]:

- Nejvýše jedno doručení – tento mód pouze odešle zprávu, není zahrnut žádný opakovací mechanismus pro případ nedoručení.
- Alespoň jedno doručení – v tomto módu je zaručeno doručení zprávy, ta však může být doručena vícekrát.
- Přesně jedno doručení – zde je ošetřeno i duplicitní doručování zpráv.

Použití sofistikovanějších metod doručení má vliv na výkon, a proto se v některých případech vyplatí zvolit nižší úroveň QoS (například při posílání idempotentních zpráv). Pro tuto práci bych však pravděpodobně zvolil záruku přesně jednoho doručení.

1.2.3.1 Šifrování a autentizace na MQTT

V této části se budu zabývat prostředky pro šifrování komunikace, které jsou dostupné v *brokeru* Mosquitto.

První možnost je pro zabezpečení komunikace využít certifikáty, podobně jako u HTTPS. Zde by se pravděpodobně také využil *self-signed* certifikát (blíže popsaný v sekci 1.2.2.1). Mosquitto navíc také vyžaduje kořenový certifikát certifikační autority [9]. Při použití *self-signed* ceritifikátů by bylo nutné tuto autoritu vytvořit a používané certifikáty u ní podepsat (pro bližší informace viz [10]). Kořenový certifikát by také bylo nutné distribuovat klientům.

Kromě certifikátů lze pro šifrování použít i PSK. V tom případě *broker* a jeho klienti používají pro zašifrování komunikace společný klíč (známý jak klientovi, tak *brokeru*). Různí klienti přitom mohou mít různé klíče. [11]

Bohužel podpora PSK v MQTT klientech není příliš rozšířená. PSK je možné použít v knihovně libmosquitto, určené pro C/C++ (s vazbami pro Python). U této knihovny se mi však podařilo najít pouze manuálovou stránku (viz [12]), bez informací o jejím dalším vývoji či udržování. Modul poskytující vazby do Pythonu byl nicméně předán projektu Paho [13].

Paho poskytuje implementace MQTT klientů pro mnoho platforem (včetně například Arduina [14]). Dokumentace klientů pro C++ a Python však možnost šifrování pomocí PSK vůbec nezmiňuje [15] [16].

Tyto možnosti lze použít i k autentizaci klientů *brokeru*. Při použití certifikátů lze v konfiguračním souboru Mosquitta zvolit `require_certificate` [11]. Poté bude od klienta vyžadován certifikát prokazující jeho totožnost. Při použití PSK lze k autentizaci využít sdílený klíč (*broker* odmítne klienty s neplatnými klíči) [11]. Kromě toho je možno použít také autentizaci pomocí uživatelského jména a hesla, která je součástí MQTT protokolu, případně klienty neověřovat vůbec (a pouze šifrovat komunikaci) [11].

1.2.4 Závěr výběru protokolu

V sekcích 1.2.2 a 1.2.3 jsem se blíže podíval na dva poměrně rozšířené protokoly aplikační vrstvy, které by bylo možné použít pro tvorbu nadřazeného systému.

Pokud by mezi požadavky na systém bylo zahrnuto zasílání nevyžádaných zpráv podřízeným systému (jak je zmíněno v sekci 1.1), zvolil bych pravděpodobně protokol MQTT. V tom je tato funkcionalita velmi snadno implementovatelná – stačí aby podřízené systémy odebíraly *topic*, na kterém by nadřazený systém publikoval zprávy.

Jelikož se však v této práci zabývám systémem, který zprávy pouze přijímá a zaznamenává, rozhodl jsem se pro HTTPS. Nasazení tohoto protokolu je o něco snazší (není potřeba na zařízení instalovat *broker* a zařizovat certifikační autoritu – stačí *self-signed* certifikát) a s jeho použitím mám více zkušeností. Také se částečně uvolní požadavky na volbu platformy (webový server bude potřeba v každém případě, při volbě HTTPS jako komunikačního protokolu mezi systémy tedy nebude nutný žádný další software).

Každopádně bude mým cílem navrhnout výslednou aplikaci tak, aby rozhraní pro podřízené systémy realizované pomocí HTTPS bylo možné snadno nahradit MQTT rozhraním.

K zabezpečení komunikace (včetně webového rozhraní) použiju *self-signed* certifikát. Hlavní důvod je požadavek na použití v místní síti, bez zaručeného přístupu k internetu. Toto rozhodnutí nemá vliv na návrh a implementaci systému, pouze na jeho nasazení – konkrétně konfiguraci webového serveru.

Pokud by provozovatel plánoval mít systém přístupný z internetu (přes registrovanou doménu), musí v každém případě k zabezpečení použít certifikát podepsaný důvěryhodnou certifikační autoritou. Pak stačí pouze v konfiguračním souboru webového serveru nahradit *self-signed* certifikát podepsaným certifikátem. Není tedy nutné provádět změny v kódu aplikace.

1.3 Ukládání dat

Zaznamenané události bude potřeba presistentně uchovávat. Zde by šly využít jednoduché textové logy, vhodnější však bude zvolit nějaký databázový systém – například kvůli širším možnostem zpracování naměřených údajů.

Z dostupných možností mě zaujal SQLite, což není klasický databázový stroj s modelem *client/server*, ale místo toho tvoří součást programu, který databázi používá. Přístup k datům je realizován pomocí přímého čtení/zápisu do databázového souboru na disku. Díky tomu má malé nároky na diskový prostor a operační paměť. [17]

Jelikož bude nadřazený systém pravděpodobně provozován na hardwaru s omezenými zdroji, představují tyto vlastnosti nezanedbatelnou výhodu. Použití SQLite také zjednouší nasazení aplikace, neboť nebude nutné vytvářet a konfigurovat databázový server.

1.4 Programovací jazyk pro tvorbu systému

Pro tvorbu systému jsem se rozhodl použít programovací jazyk Python. S tímto jazykem mám nejvíce zkušeností co se týče implementace webových aplikací. Je také dostatečně rozšířený, takže výsledný systém bude možné nasadit na poměrně širokém spektru platforem bez nutnosti složitějšího portování.

K vytvoření webového rozhraní i API pro podřízené systémy jsem zvolil framework Flask. Hlavní důvod jsou opět předchozí zkušenosti s tímto frameworkem. Flask také dává více volnosti při návrhu aplikace než například také velmi rozšířený framework Django.

1.5 Výběr platformy

Pro realizaci systému je nutné zvolit vhodnou platformu. Jelikož je cílem práce vytvořit fyzické zařízení, rozhodl jsem se jako základ použít některý z jednodeskových počítačů, které jsou v dnešní době na trhu. Tyto počítače bývají cenově velmi dostupné a zároveň poskytují dostatečný výkon a podporu pro provoz systému.

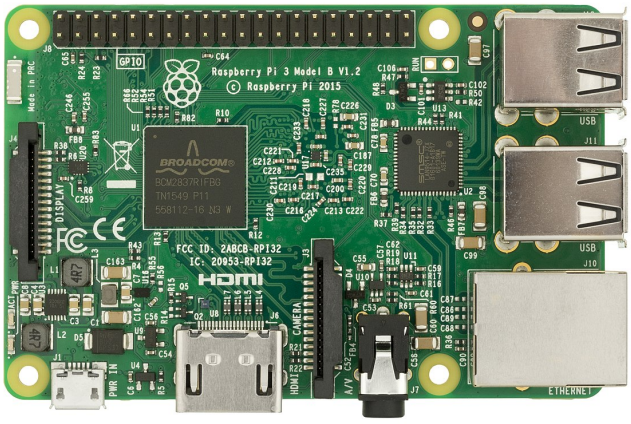
Při výběru počítače byla nejdůležitejším kritériem podpora softwaru potřebného k implementaci monitorovacího systému. Na základě předchozí analýzy je tedy vyžadován následující software:

- Webový server Apache2.
- Databázový systém SQLite.
- Programovací jazyk Python 3.
  - Webový framework Flask.

Pro provoz tohoto softwaru bude potřeba plnohodnotný operační systém, což vylučuje platformy využívající jednoduché mikrokontroléry, jako například Arduino Uno. Kromě toho je nutné připojení k síti pomocí Ethernetu nebo WiFi.

V dalších sekcích jsem se blíže podíval na jednodeskové počítače Raspberry Pi (sekce 1.5.1) a Zybo Zynq-7000 (sekce 1.5.2) a zvážil jejich výhody a nevýhody pro implementaci systému.

1.5.1 Raspberry Pi



Obrázek 1.3: Raspberry Pi 3 (obrázek převzat z `https://en.wikipedia.org/wiki/Raspberry_Pi`)

Raspberry Pi je velmi rozšířený jednodeskový počítač. Jeho poslední verze, Raspberry Pi 3, je postavena na SoC Broadcom BCM2837 s čtyřjádrovým procesorem ARM Cortex A53, který je až o 50 % rychlejší než procesor předchozí verze [18].

Dále nová verze přináší vlastní WiFi modul [18], není tedy nutné se spoléhat na externí moduly. Kromě toho je možné počítač připojit k síti pomocí Ethernetového portu. Ten je omezený na 100 Mb/s [18], to by však vzhledem k objemu dat přenášených mezi nadřazeným a podřízenými systémy nemělo představovat problém.

Deska také obsahuje čtyři USB a jeden HDMI port. Ty nejsou pro implementovaný systém zásadní, nicméně při počáteční konfiguraci zařízení (například nastavení WiFi hesla) může být připojení monitoru a klávesnice pro některé uživatele pohodlnější než použití SSH či sériové linky. Připojený monitor se také hodí při řešení problémů se startem operačního systému.

Raspberry Pi 3 bohužel nemá vlastní bateriově zálohovaný RTC obvod a k udržování času využívá NTP [19]. K tomu je však zapotřebí internetové

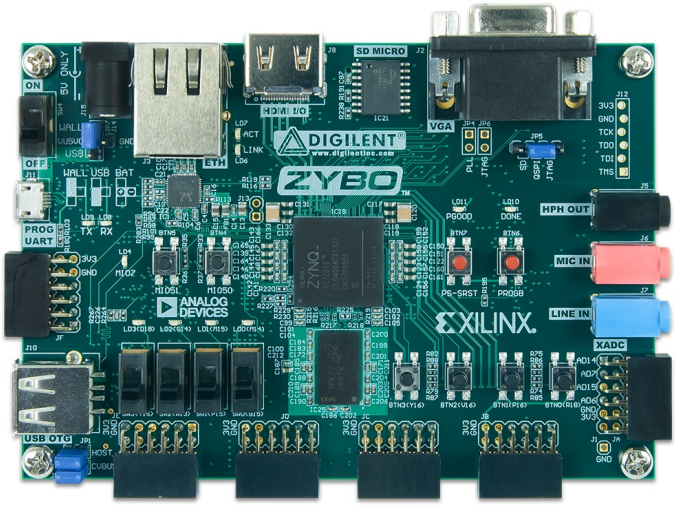
připojení. Jelikož by zařízení mělo být možné používat i v síti bez přístupu k internetu, je nutné připojit externí RTC obvod, například pomocí I2C sběrnice. Poté je možné systémové hodiny synchronizovat offline pomocí tohoto obvodu (místo NTP).

S počítačem je možné použít množství operačních systémů, z nichž nejrozšířenější je pravděpodobně Raspbian, linuxový systém postavený na Debianu. Pro ten jsou dostupné všchny potřebné softwarové balíčky popsané v sekci 1.5. Jelikož počítač nemá žádné vlastní úložiště, je nutné operační systém provozovat na vložené SD kartě.

Jednou z výhod tohoto počítače je obrovské množství podporovaných hardwarových periferií a knihoven pro ně. V této práci pravděpodobně využiju pouze zmíněný RTC obvod, případné další rozšiřování systému (například o vestavěný LCD displej) bude na Raspberry Pi pravděpodobně snadnější než na jiných platformách.

Kromě široké podpory je hlavní výhodou Raspberry Pi jeho cena. Poslední verze se pohybuje kolem 1200 Kč. K celkovým nákladům na systém je ještě třeba připočítat cenu RTC obvodu a SD karty. Zde počítám s použitím již připraveného modulu s obvodem PCF8523. Ten vyjde asi na 200 Kč. Jako úložiště by měla plně dostačovat 16GB SD karta, která se dá pořídit za 200 Kč. Celkové náklady na hardware systému se tedy měly pohybovat kolem 1600 Kč.

1.5.2 Zybo Zynq-7000



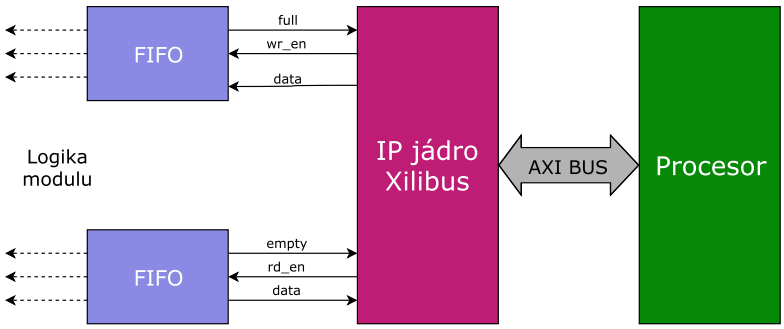
Obrázek 1.4: Připravek Zybo Zynq-7000 (obrázek převzat z `https://www.xilinx.com/products/boards-and-kits/1-4azfte.html`)

Tento přípravek od společnosti Digilent je postavený na SoC Xilinx Zynq

Z-7010. Hlavní předností tohoto čipu je kombinace dvoujádrového procesoru ARM Cortex A9 s FPGA odpovídající sérii Artix-7 [20].

Díky tomu je možná těsná integraci mezi aplikací běžící na procesoru a výkonnými, úzce specializovanými moduly, které jsou syntetizované na FPGA. Tento přístup, kdy je hardware a software systému vyvíjen souběžně se označuje jako *hardware/software codesign*.

Pro SoC ze série Zynq vznikla linuxová distribuce Xilinux, vycházející z Ubuntu. Kromě plnohodnotného operačního systému (včetně například grafického rozhraní) poskytuje Xilinux také ovladače pro komunikaci s FPGA pomocí AXI sběrnice [21]. K tomu využívá IP jádro Xilibus, které funguje jako adaptér mezi procesorem a FPGA modulem (viz obrázek 1.5). Ten pak ke komunikaci může využívat standardni FIFO fronty a nemusí se zabývat AXI sběrnicí [21].



Obrázek 1.5: Blokové schéma využití IP jádra Xilibus [21]

Jelikož Xilinux staví na Ubuntu (konkrétně na verzi 12.04 LTS [21]), neměl by být problém nainstalovat software potřebný pro provoz nadřazeného systému (viz sekce 1.5).

Přípravek je možné připojiř k síti pomocí Ethernetového portu, který podporuje rychlost až 1Gb/s [20]. WiFi připojení by bylo možné realizovat pomocí USB modulu. Dále přípravek obsahuje HDMI a VGA port, audio konektory, čtyři tlačítka, čtyři přepínače a slot pro SD kartu [20].

Na přípravku je také k dispozici 128 MB flash paměti [20], k provozu tedy teoreticky není potřeba SD karta. Tato paměť by však pravděpodobně nestačila k instalaci vhodného operačního systému a potřebného softwaru. I zde by tedy bylo nutné použít SD kartu.

Stejně jako Raspberry Pi tato deska postrádá RTC obvod. Firma Digilent však dodává externí bateriově zálohované hodiny, které lze připojit pomocí Pmod rozhraní [22].

Nevýhodu této desky (především v porovnání Raspberry Pi) je její cena. Ta se pohybuje okolo 4000 Kč. K tomu je nutné přičíst náklady na SD kartu a RTC obvod, případně i WiFi modul. Cena celého zařízení by se tedy pohybovala v rozmezí 4500 až 5000 Kč.

1.5.3 Závěr výberu platformy

	Raspberry Pi 3	**Zybo Zynq-7000**
SoC	Broadcom BCM2837	Xilinx Zynq Z-7010
Procesor	ARM Cortex A53	ARM Cortex A9
	4 jádra, 1,2 GHz	2 jádra, 650 MHz
RAM	1024 MB LPDDR2	512 MB DDR3
FPGA	–	ekvivalent řady Artix-7
Úložiště	SD karta	128 MB Flash, SD karta
Síť	100 Mb/s Ethernet, WiFi	až 1 Gb/s Ethernet
Cena	1200 Kč	4000 Kč

Tabulka 1.1: Srovnání platforem Raspberry Pi 3 a Zybo Zynq-7000

[18] [20]

Jak vyplívá z tabulky 1.1, Raspberry Pi 3 poskytuje znatelně výkonnější procesor a více paměti za méně než třetinu ceny desky Zybo. Hlavní přidaná hodnota této platformy tedy spočívá v integraci s FPGA, ta má však v případě této práce pouze velmi omezené využití.

FPGA by bylo možné využít například k šifrování úložiště, vzhledem k předpokládáným objemům dat by však zrychlení oproti softwarovému šifrování neospravedlnilo vysokou cenu přípravku. Na druhou stranu vyšší výkon procesoru u Raspberry Pi může mít pro nadřazený systém význam, například z hlediska odezvy uživatelského rozhraní.

Jako platformu pro implementaci nadřazeného systému jsem tedy zvolil Raspberry Pi 3, především kvůli příznivé ceně, výrazně lepšímu poměru cena/výkon (pro tuto práci), a také kvůli podpoře a rozšiřitelnosti.

1.5.4 Provoz aplikace na cloudové platformě

V této části bych chtěl popsat alternativu k provozu systému na dedikovaném zařízení, a to možnost využít virtuální server na některé cloudové platformě. Primární cíl práce je sice vytvořit nadřazený systém jako jednoúčelové zařízení (postavené na Raspberry Pi), nicméně provoz výsledné aplikace v cloudu může být v určitých situacích vhodnější řešení.

Jedním z možných uplatnění této varianty je monitorování více garážových komplexů. Místo lokálního nadřazeného systému by mohly všechny podřízené systémy z každého komplexu komunikovat s jedním globálním systémem, provozovaným na virtuálním serveru a dostupným z internetu, jak je naznačeno na obrázku 1.5.4. Při tomto provozu je však nutnost mít registrovanou doménu a zabezpečit spojení podepsaným certifikátem.

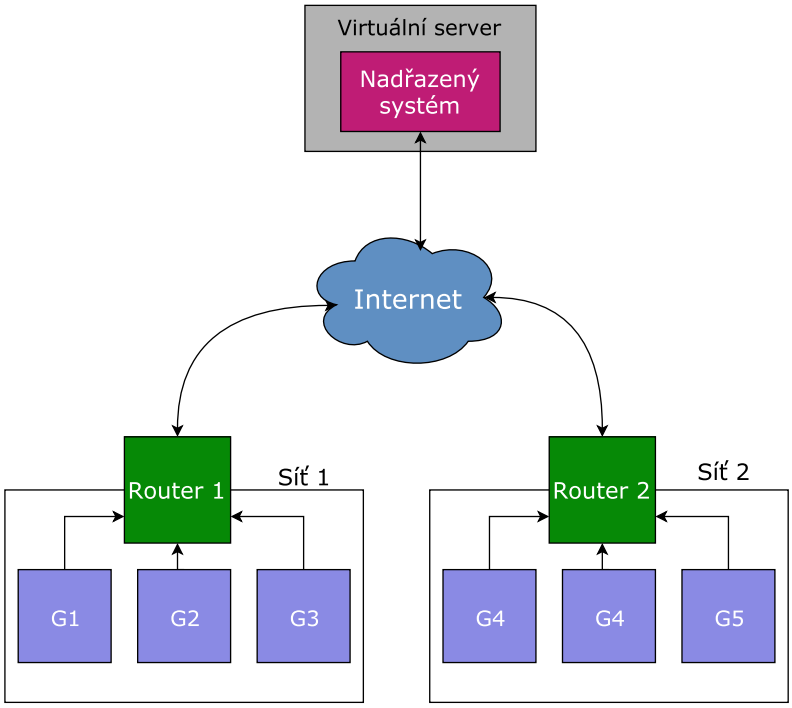
Virtuální servery nabízí například firma DigitalOcean. Cena serveru závisí na počtu výpočetních jader, dostupně RAM a velikosti úložiště. Nejlevnější konfigurace přijde na 5 dolarů měsíčně a nabízí jednojádrový procesor, 1 GB

RAM a 25 GB SSD [23]. Předpokládám, že tento výkon by stačil pro základní provoz systému, v případě vyššího počtu podřízených systémů je však možnost zvyšovat dostupnou RAM a jádra CPU.

S těmito virtuálními servery lze použít řadu běžných linuxových operačních systémů jako Ubuntu, Debian či Fedora [24], dostupnost softwaru potřebného pro spuštění aplikace tedy není problém.

Hlavní nevýhodou tohoto přístupu je poněkud složitější nasazení a spuštění systému. Registrace domény, získání certifikátu a základní konfigurace webového serveru se dá sice částečně automatizovat, pro většinu uživatelů bude však pravděpodobně snazší použít již připravené Raspberry Pi.

Další problém může představovat hlavní výhoda tohoto řešení, a to přístupnost serveru z internetu. Ta významně zvyšuje *attack surface* celého nadřazeného systému, zvlášť oproti alternativě využívající k propojení systémů pouze Ethernet u kterého má (na rozdíl od WiFi) provozovatel fyzický přehled o připojených zařízeních.



Obrázek 1.6: Struktura systému provozovaného na virtuálním serveru

Kapitola 2

Návrh

2.1 Návrhový vzor MVC

Struktura nadřazeného systému je vhodná k použití návrhového vzoru MVC, tedy *model-view-controller*. *Model* zde představují garáže (podřízené systémy), k nim vázané události a logika jejich vyhodnocování.

View je zobrazení těchto dat, tedy především generované HTML stránky webového rozhraní. Jako další *view* je možné považovat získávání dat (například ve formátu JSON) pomocí API nadřazeného systému, třeba při zasílání registračních klíčů podřízeným systémům.

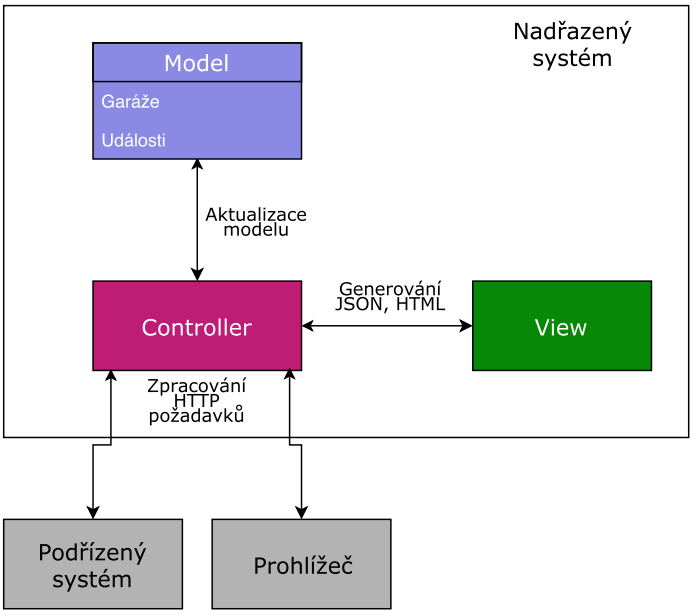
Controller je pak část aplikace, která se stará o zpracování HTTP požadavků. Ty mohou přicházet jednak z uživateloval prohlížeče, jednak od podřízených systémů. Na základě těchto požadavků pak *controller* posílá příslušné příkazy *modelu*. Struktura aplikace při použití vzoru MVC je naznačena na obrázku 2.1.

Hlavní motivací pro použití tohoto vzoru je snadná rozšiřitelnost. Pokud by například bylo potřeba aplikaci doplnit o komunikaci s podřízenými systémy pomocí MQTT, stačí pouze vytvořit vhodný *controller*. Ten pak může využívat *model* aplikace stejným způsobem jako HTTP *controller*.

2.2 Flask Blueprints

ta sekce s blueprintama bude asi nejlepsi tady hned z kraje
`http://flask.pocoo.org/docs/0.12/blueprints/`
rekneme ze budeme mit tri blueprinty (moduly):

- main modul – tady bude definovanej ten hlavni model, tj garaze, eventy a pripadne naka fasada na tim. Controller a view pak bude zprostredkovavat uzivatelsky rozhrani (krome loginu) webovy stranky. V timhle rozhrani pude proklikavat ty garaze a eventy, zapinat registraci mod a vytvaret/mazat garaze. To vytvareni garazi je hlavne kvuli nakejm dev



Obrázek 2.1: Struktura MVC aplikace

options, primarne se budou garaze vytvaret skrz to api (tj tim cudlikem na podrizenym systemu). Vytvorit garaz v uzivatelskym rozhrani nebude vyzadovat zapnutej registracni mod, tj to bude uplne jinej pozadavek na uplnej jinej controller – ten v main modulu a ne v api modulu

- api modul – modul co bude zprostredkovavat api pro podrizeny systemy. tj tady se nebude generovat zadny html nebo veci pro uzivatelsky rozhrani, ale ciste jen zpracovavat pozadavky vod podrizenejch systemu. Modul bude vyuzivat tu fasadu z main modulu pro pristup k databazi (stejne jako main modul). Controller v timhle modulu bude teda resit pozadavky na vytvareni novejch garazi pomoci API. To jestli je zapnutej registracni mod bude resit ten model, v tim bude vodlisna funkce pro vytvoreni garaze pres reg. mod a pres rozhrani a prislusny controllery budou volat prislusnou funkci. Krome toho tenhle controller bude resit pozadavky na vytvoreni eventu.

- auth modul – tenhle modul bude resit ciste prihlaseni uzivatele do webovyho rozhrani

Z toho vypliva ze jadrem ty aplikace bude ten datovej model (garaze, eventy atd...) a fasada nad nim. Ta fasada by teda mela umet nasledujici veci (za pomlckou kterej modul to bude pouzivat):

- vytvorit garaz (pozadavek z web. rozhrani) – main

- vytvorit garaz (pozadavek z api – tj kontrola reg. modu) – api
- vypnout/zapnout reg. mod – main
- smazat garaz – main
- vratit vsechny garaze – main
- vratit konkretni garaz – main, (api?)
- vratit vsechny eventy – main
- vratit eventy ke garazi – main
- vytvorit event vazanej ke garazi – api
- kontrola klicu pozadavku vod api – api
- vnitrni udrzovani stavu garazi a vyhodnocovani udalosti – main, api

ten model by mozna nemel bejt cast zadnyho toho modulu (blueprintu) ale bejt zvlast, kdyz ho budou pouzivat dva moduly najednou. Ze by se instancioval v tim hlavnim init.py souboru podobne jako se tam instanciuje ta databaze

2.3 Model

pouziti sqlalchemy – to resit az v implementaci

2.3.1 Garáž

2.3.1.1 Stav garáže

2.3.2 Událost

2.3.2.1 Vyhodnocení události

2.3.3 Fasáda

2.4 Controller

Flask API

2.5 View

2.6 Autentizace

2.6.1 Autentizace uživatele

2.6.2 Autentizace podřízeného systému

2.6.2.1 Registrační mód

2.6.3 API

Kapitola 3

Implementace

pouziti sqlalchemy (http://flask-sqlalchemy.pocoo.org/2.3/quickstart/#a-minimal-application)

pouziti flask blueprintu (http://flask.pocoo.org/docs/0.12/blueprints/) – tohle mozna probrat uz v narvrhu – jaky budem mit moduly a tak

zakladni struktura aplikace zalozena na https://www.digitalocean.com/community/tutorials/how-to-structure-large-flask-applications#working-with-modules-and-blueprints-(components)

```
# ... code here ...

import numpy as np

def foo(a):
    print(a)

class FooBar:
    def __init__(self):
        self.b = 10

a = [1, 2, 3, 4]
for i in a:
    if i == 2:
        print("hello world")
```

Ukázka 2: Testovací listing

Kapitola 4

Nasazení

tady bude neco vo nasazovani na RPI (tj rozjet apache, vygenerovat certifikaty atd., viz `https://github.com/ggljzr/mi-dip-impl/tree/master/deployment`)

Kapitola 5

Testování

Závěr

Literatura

[1] Arduino: Web Client. 2015, [cit. 2017-10-25]. Dostupné z: https://www.arduino.cc/en/Tutorial/WebClient

[2] Grokhotkov, I.: esp8266wifi – Client Example. 2017, [cit. 2017-10-25]. Dostupné z: https://github.com/esp8266/Arduino/blob/master/doc/esp8266wifi/client-examples.rst

[3] Electronic Frontier Foundation: Certbot – About. [cit. 2017-10-18]. Dostupné z: https://certbot.eff.org/about/

[4] Electronic Frontier Foundation: Let's Encrypt – Frequently Asked Questions. 2017, [cit. 2017-11-07]. Dostupné z: https://letsencrypt.org/docs/faq/

[5] Wallen, J.: When are self-signed certificates acceptable for businesses? 2017, [cit. 2017-11-08]. Dostupné z: https://www.techrepublic.com/article/when-are-self-signed-certificates-acceptable-for-businesses/

[6] Lampkin, V.: What is MQTT and how does it work with WebSphere MQ? 2012, [cit. 2017-10-25]. Dostupné z: https://www.ibm.com/developerworks/mydeveloperworks/blogs/aimsupport/entry/what_is_mqtt_and_how_does_it_work_with_websphere_mq?lang=en

[7] Jaffey, T.: MQTT and CoAP, IoT Protocols. 2014, [cit. 2017-11-12]. Dostupné z: https://eclipse.org/community/eclipse_newsletter/2014/february/article2.php

[8] Newsom, C.: Mosquitto Message Broker. 2016, [cit. 2017-11-16]. Dostupné z: https://github.com/mqtt/mqtt.github.io/wiki/Mosquitto-Message-Broker

[9] Light, R.: mosquitto.tls – Mosquitto Manual. [cit. 2017-11-20]. Dostupné z: https://mosquitto.org/man/mosquitto-tls-7.html

[10] Nguyen, J.: OpenSSL Certificate Authority. 2015, [cit. 2017-11-20]. Dostupné z: https://jamielinux.com/docs/openssl-certificate-authority/

[11] Light, R.: mosquitto.conf – Mosquitto Manual. [cit. 2017-11-20]. Dostupné z: https://mosquitto.org/man/mosquitto-conf-5.html

[12] Light, R.: libmosquitto – Mosquitto Manual. [cit. 2017-11-21]. Dostupné z: https://mosquitto.org/man/libmosquitto-3.html

[13] Eclipse Foundation: Python – Mosquitto Documentation. [cit. 2017-11-21]. Dostupné z: https://mosquitto.org/documentation/python/

[14] Eclipse Foundation: Embedded MQTT C/C++ Client Libraries. [cit. 2017-11-21]. Dostupné z: http://www.eclipse.org/paho/clients/c/embedded/

[15] Eclipse Foundation: Paho C++ Documentation. [cit. 2017-11-21]. Dostupné z: http://www.eclipse.org/paho/files/cppdoc/index.html

[16] Eclipse Foundation: Paho Python Documentation. [cit. 2017-11-21]. Dostupné z: https://pypi.python.org/pypi/paho-mqtt

[17] About SQLite. [cit. 2017-12-12]. Dostupné z: https://www.sqlite.org/about.html

[18] Benchoff, B.: Introducing the Raspberry Pi 3. 2016, [cit. 2018-01-25]. Dostupné z: https://hackaday.com/2016/02/28/introducing-the-raspberry-pi-3/

[19] Adafruit: Adding a Real Time Clock to Raspberry Pi. 2016, [cit. 2018-01-25]. Dostupné z: https://learn.adafruit.com/adding-a-real-time-clock-to-raspberry-pi/overview

[20] Digilent: ZYBO FPGA Board Reference Manual. 2016, [cit. 2018-01-29]. Dostupné z: https://reference.digilentinc.com/_media/zybo:zybo_rm.pdf

[21] Xilibus: Getting Started with Xilinux for Zynq-7000. [cit. 2018-01-29]. Dostupné z: http://xillybus.com/downloads/doc/xillybus_getting_started_zynq.pdf

[22] Digilent: Pmod RTCC Reference Manual. 2016, [cit. 2018-01-30]. Dostupné z: https://reference.digilentinc.com/_media/reference/pmod/pmodrtcc/pmodrtcc_rm.pdf

[23] DigitalOcean: DigitalOcean – Pricing. 2018, [cit. 2018-02-05]. Dostupné z: https://www.digitalocean.com/pricing/

[24] DigitalOcean: DigitalOcean – Droplets. 2018, [cit. 2018-02-05]. Dostupné z: https://www.digitalocean.com/products/droplets/

Příloha **A**

Seznam použitých zkratek

API

AXI

CPU

DDR

EEPROM

FPGA

HDMI

HTML

HTTP Graphical user interface

HTTPS Graphical user interface

I2C

IP Ip jadro

JSON

LCD

LPDDR

LTS

MQTT Graphical user interface

MVC

NTP Network time protocol

OSI

PSK

QoS

RAM

RTC

SD

SSD

SoC

TCP/IP Graphical user interface

URL

USB

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS