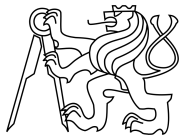
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Zařízení pro získávání a zpracování dat z internetu na platformě Raspberry Pi
Student:	Bc. Jakub Kužel
Vedoucí:	Ing. Pavel Kubalík, Ph.D.
Studijní program:	Informatika
Studijní obor:	Počítačové systémy a sítě
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Z počítače Raspberry Pi vytvořte zařízení pro zpracování a prezentaci dat z internetu.
Pro toto zařízení vyberte vhodný OS.
Navrhněte a naprogramujte aplikaci v jazyce C++ pro získávání a zpracování dat z různých zdrojů (periferie Raspberry Pi, SD karta, internet).
Za účelem získání dat ze souboru nebo webové stránky použijte jednoduchý skriptovací jazyk.
Zařízení bude možné konfigurovat a ovládat pomocí zabezpečeného síťového spojení ze vzdáleného PC nebo přímo lokálně pomocí dotykového displeje.
Aplikace bude umožňovat vzdálenou instalaci nového digitálního certifikátu pro zabezpečený přenos dat.
Výsledné řešení otestujte.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 29. ledna 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Diplomová práce

Zařízení pro získávání a zpracování dat z internetu na platformě Raspberry Pi

Bc. Jakub Kužel

Vedoucí práce: Ing. Pavel Kubalík, Ph.D.

9. května 2017

Poděkování

Chtěl bych poděkovat Ing. Pavlovi Kubalíkovi, Ph.D. za vedení mé diplomové práce a cenné rady. Děkuji také své rodině za podporu, kterou mi poskytli při zpracování této 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 9. května 2017 .....

Odkaz na tuto práci

Kužel, Jakub. *Zařízení pro získávání a zpracování dat z internetu na platformě Raspberry Pi.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Hlavím cílem této diplomové práce je prověřit možnosti počítače Raspberry Pi jako nástroje pro prezentaci dat z internetu. Vzniklá aplikace spravuje uživatelské skripty, které slouží k prezentaci dat prostřednictvím připojeného dotykového displeje nebo webového rozhraní zařízení. Práce je rozdělena na dvě hlavní části. V analytické části se věnuji rozboru možností a návrhu řešení. V implementační části je popsán průběh vývoje zařízení.

Klíčová slova Raspberry Pi, prezentace dat, dotykový displej, webové rozhraní, uživatelský skript, message broker, ZeroMQ

Abstract

The main goal of this diploma thesis is to examine the possibilities of Raspberry Pi computer as the tool for presentation data from the internet. The final aplication manages user scripts. This scripts are designet to present data using the attached display or web interface. The thesis is devided into two main parts. In the analytical part is described possibilities and design of the solution. The implementation part describes the development of the device.

Keywords Raspberry Pi, data presentation, touch-screen, web interface, user script, message broker, ZeroMQ

Obsah

Seznam obrázků

Úvod

Cílem této diplomové práce je vytvořit zařízení, sloužící jako nástroj k prezentaci dat dle vlastních požadavků. K tomuto účelu bude přizpůsoben počítač Raspberry Pi s připojeným dotykovým displejem. Smyslem práce je prověřit a prakticky vyzkoušet, jakým způsobem by bylo možné uživateli umožnit vytvářet vlastní prezentace pomocí jednoduchého skriptu.

Aby mělo cílové zařízení smysl, bude umožňovat jednoduchou správu libovolného počtu uživatelských skriptů. Jedním z cílů práce je tedy navrhnout a vytvořit takové ovládací rozhraní, pomocí kterého bude možné skripty do zařízení nahrávat a dále je udržovat.

K počítači byl zakoupen jednoduchý 3.5″ dotykový displej KeDei. Tento bude sloužit k výběru z nahraných skriptů a k prezentaci jejich výstupu. Pro displej bude tedy nutné navrhnout a vytvořit aplikaci, která jej bude obsluhovat. V rámci toho je nutné navrhnout způsob, jakým bude aplikace s výstupem skriptu pracovat tak, aby bylo možné vytvořit přehlednou prezentaci.

Cílové zařízení by mělo tvořit ucelený nástroj, který bude připraven fungovat sám po startu zařízení bez nutnosti zásahu uživatelem. Součástí práce tedy bude i vhodná konfigurace systému a vzniklých aplikací.

Práce je rozdělena do několika kapitol. Kromě úvodu se nejprve věnuji detailnímu popisu zadání a požadavkům na cílové zařízení. V další kapitole se věnuji rešerši dalších aplikací, určených pro počítač Raspberry Pi a také rešerši podobných zařízení jako je počítač Raspberry Pi. Třetí kapitola je určena pro analýzu a návrh řešení celého zařízení. Tato kapitola je věnována jednak analýze vytvářené aplikace a dále návrhu tvorby uživatelských skriptů. Ve čtvrté kapitole je popsána samotná realizace projektu. Poslední kapitola se věnuje demonstraci a otestování aplikace na několika ukázkových příkladech.

Kapitola 1

Specifikace zadání

Úkolem této práce je vytvořit z počítače Raspberry Pi zařízení pro zpracování a prezentaci dat z internetu. K zařízení bude připojený dotykový displej, který bude sloužit k samotné prezentaci dat. Smyslem zařízení bude umožnit uživateli tvořit vlastní prezentace pomocí jednoduchých skriptů, které budou prostřednictvím konfiguračního rozhraní nahrávány na zařízení. Úkolem skriptu bude stáhnout data z internetu a zpracovat je dle vlastních potřeb. Zařízení bude tedy tvořit platformu, která bude uživatelské skripty obsluhovat a prezentovat jejich výstup.

1.1 Cíle práce

Ze zadání plyne několik následujících úkolů, kterým se v této práci budu věnovat.

- Výběr vhodného operačního systému
- Vývoj aplikace pro získávání a zpracování dat z různých zdrojů a jejich prezentace na připojeném dotykovém displeji
- Návrh uživatelských skriptů jako nástroje pro zpracování získaných dat
- Tvorba nástroje pro vzdálenou konfiguraci zařízení (primárně nahrávání skriptů).
- Zabezpečení komunikace se zařízením pomocí digitálního certifikátu
- Testování funkčnosti výsledného řešení

Ačkoliv to není v zadání práce přímo řečeno, bude zařízení umožňovat prezentaci dat, kromě samotného displeje také prostřednictvím webového rozhraní. Výsledkem práce by mělo být tedy kompletní zařízení zahrnující jak programovou výbavu tak i veškerou potřebnou konfiguraci.

Kapitola 2

Rešerše

Počítač Raspberry Pi je často využíván k ovládání různých zařízení, ale lze ho využít i k přehrávaní multimédií, přístupu k internetu apod. Původně byl počítač Raspberry Pi vyvinut jako studijní pomůcka s cílem podpořit výuku informatiky. Obvyklá aplikace tohoto zařízení slouží spíše k soukromým účelům než aby na něm byla založena komerční řešení. Proto také existuje na internetu celá řada návodů jak počítač Raspberry Pi ke konkrétnímu účelu přizpůsobit. Nicméně existují i některá komerční řešení, založená na počítači Raspberry Pi. Například multimediální přehrávač Slice, či digitální kamera OTTO, která tvoří nahrávky formou GIF animací.

V této kapitole bych se rád věnoval rozboru podobných řešení, založených především na platformě Raspberry Pi. Nicméně vzhledem k tomu, že zadání práce je značně specifické, je obtížné obdobná řešení najít. Proto se zde zaměřuji na další, alespoň částečně podobné aplikace.

2.1 Aplikace pro počítač Raspberry Pi

Účelem této práce je, stručně řečeno, vytvořit zařízení pro prezentaci dat z internetu. Aplikace se zaměřuje především na data získána prostřednictvím protokolu HTTP. Těmito daty může být tedy například webová stránka, ale i libovolné API, které zpřostředkovává data právě prostřednitvím protokolu HTTP. Příkladem využití takového API je ukázková aplikace, která zobrazuje aktuální předpověď počasí. Tato ukázková aplikace je popsána v sekci 5.2.

V tomto kontextu můžeme tedy porovnat práci s dalšími aplikacemi počítače Raspberry Pi, sloužícími k podobnému účelu. Jednou z takových aplikací je projekt RPI Weather. RPI Weather je aplikace určená pro Raspberry Pi, konkrétně pro model A. Nejedná se o hotové zařízení, ale o softwarovou aplikaci, určenou k jeho ovládání. Aplikace je určená konkrétně k ovládání LED displeje (mřížka 8x8 LED), na kterém je zobrazena předpověď počasí na následující čtyři dny formou ikony. Toto zařízení je zobrazeno na obrázku 2.1.

Obrázek 2.1: Ukázka projektu RPI Weather [1]

Aplikace čerpá data prostřednictvím několika veřejně dostupných API. Jedním z nich je i api.openweathermap.org, stejný zdroj využívá i ukázková aplikace. Projekt RPI Weather je veřejně dostupný na githubu na adrese `https://github.com/caternuson/rpi-weather`. [1]

Velké množství aplikací pro počítač Raspberry Pi, které jsou na internetu k dohledání, jsou často spíše instrukce a návody, jak aplikaci zrekonstruovat, než hotová řešení.

Jednou z takových aplikací je modifikace zařízení pro vytvoření interaktivního nástěnného kalendáře. Jde o postup konfigurace zařízení popisující, jak pomocí počítače Raspberry Pi a připojeného monitoru kalendář vytvořit. Řešení lze vidět na obrázku 2.2. Řešení je postaveno na prohlížeči Iceweasel, kde je jednoduše zobrazena stránka s google kalendářem. Popis postupu je možné najít na stránce
`http://www.instructables.com/id/Raspberry-Pi-Wall-Mounted-Google-Calendar/`. [2]

Poměrně běžným využitím počítače Raspberry Pi je také obsluha různých připojených zařízení. Existuje velké množství různých zařízení a rozšiřujících modulů, určených výhradně pro připojení k Raspberry Pi. Patří mezi ně i řada různých senzorů, jako je měřič teploty, tlaku, vlhkosti, ale i například senzor

Obrázek 2.2: Raspberry Pi Wall Mounted Google Calendar [2]

pohybu. Obvyklou aplikací Raspberry Pi je obsluha právě těchto senzorů, tedy sběr údajů a jejich prezentace. Zde se opět jedná spíše o řadu návodů a nápadů jak takovou aplikaci zprovoznit spíše, než o hotová řešení, což je pro Raspberry Pi typické.

2.2 Smartphone a tablet

V určitém ohledu můžeme práci srovnávat i s chytrými mobilními telefony či tablety. Tato zařízení samozřejmě nabízejí daleko více možností, nicméně existuje řada užitečných aplikací, které můžeme s touto srovnávat. Zde mám na mysli zejména aplikace, které jsou také určeny k prezentaci různých „užitečných údajů“. Tím myslím například aplikace, které uživateli přehlednou formou nabízejí například program kina, TV program, jízdní řády veřejné dopravy či již zmíněnou předpověď počasí. Takových aplikací je celá řada a mnohé z nich také umožňují umístit na hlavní obrazovku mobilního zařízení widget s přehledem těch nejdůležitějších údajů. Tím tyto aplikace uživateli ještě více ulehčí přístup k informacím, které potřebuje. Možnosti těchto aplikací jsou jistě daleko větší než to, čeho je možno dosáhnout tvorbou uživatelských skriptů v této práci. Tato práce si ani neklade za cíl konkurovat mobilním aplikacím. Účel práce spočívá právě v možnosti vlastní tvorby uživatelských skriptů, což umožňuje výslednou prezentaci vytvořit zcela podle vlastních představ.

2.3 Rešerše obdobných zařízení

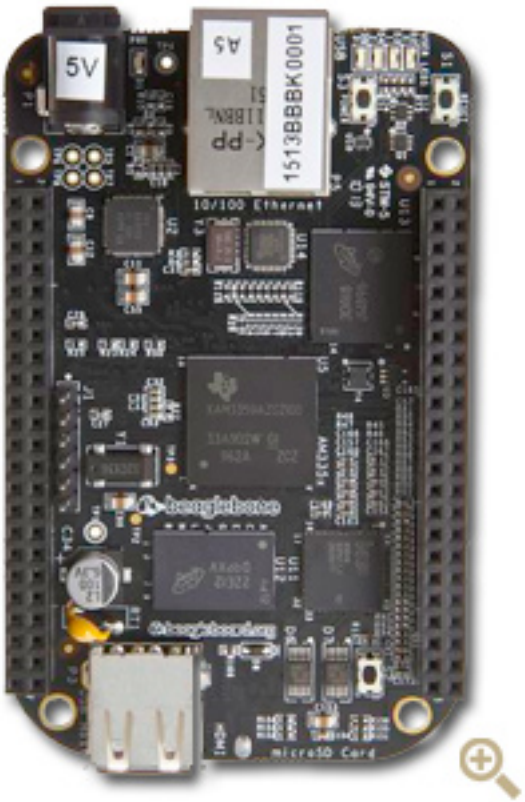
Dle zadání práce bude vývoj probíhat na počítači Raspberry Pi. Raspberry Pi je pravděpodobně nejznámější single-board miniaturní počítač, nicméně není

zdaleka jediný. Existuje celá řada podobných zařízení, která by zde mohla být použita. V této části se budu věnovat právě rešerši těchto alternativ. Nicméně vzhledem k jejich počtu není prostor věnovat pozornost všem. Proto se v následujících sekcích zaměřím pouze na několik vybraných, více či méně známých, zařízení.

2.3.1 BeagleBone Black

BeagleBone Black je zatím poslední zařízení z rodiny single-board počítačů BeagleBoard. BeagleBoard Black obsahuje 1GHz procesor AM335x ARM Cortex-A8 a 512MB DDR3 operační paměti. Zařízení je vybaveno jedním USB a HDMI portem a také ethernet konektorem. [3]

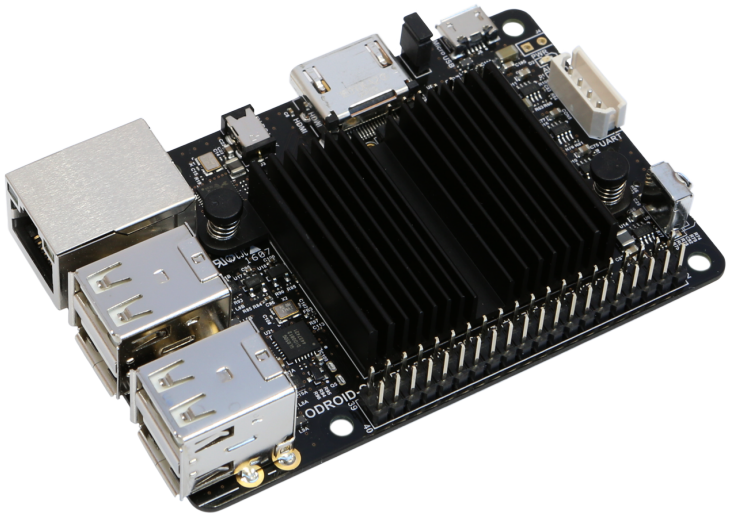
Zařízení podporuje řadu převážně Linuxových systémů. Mezi ně patří například Debian, Ubuntu, ale i systém Android. Pro počítač BeagleBone Black byl sesteven image systému Android 4.2.2. Jelly Bean. [4]



Obrázek 2.3: Počítač BeagleBone Black [3]

2.3.2 ODROID

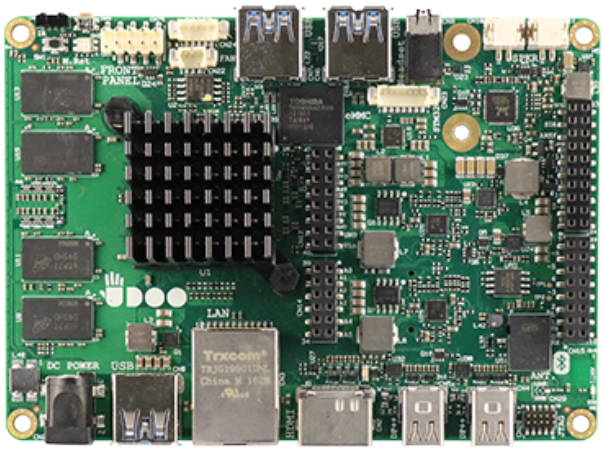
ODROID je serie single-board počítaču, založených na architektuře ARM. Zatím poslední verze tohoto počítače vyšla v roce 2016 a jedná se o model ODROID-C2. ODROID-C2 obsahuje 64bit, 1.5GHz, 4 jádrový procesor Cortex-A53 a 2GB DDR3 operační paměti. [5] Počítač Raspberry Pi 3 (což je zatím poslední model počítače Raspberry Pi) pro porovnání obsahuje 64bit, 4 jádrový, 1.2Ghz procesor a pouze 1GB operační paměti. [6] Počítače ODROID také podorují řadu operačních systémů. Na stránkách výrobce (`http://odroid.com/dokuwiki/doku.php?id=en:odroid-c2`) lze dohledat, že je zařízení kompatibilní především s Linuxovými operačními systémy. Na této stránce jsou přednostně zmíněny systémy Ubuntu a Android.



Obrázek 2.4: Počítač ODROID-C2 [5]

2.3.3 UDOO

UDOO je další ze zástupců řady single-board počítačů. UDOO je vlastně kombinací dvou platforem. Například model UDOO X86 kombinuje platformu s procesorem intel a Arduino 101 kompatibilní platformou. Díky tomu je možné na tomto zařízení spouštět veškerý software dostupný pro běžné PC a stejně tak software dostupný pro Arduino 101 platformu. Vzhledem k použité platformě je možné na zařízení použít veškeré x86 kompatibilní operační systémy Linux, Windows ale i systém Android. [7]



Obrázek 2.5: Počítač UDOO X86 [8]

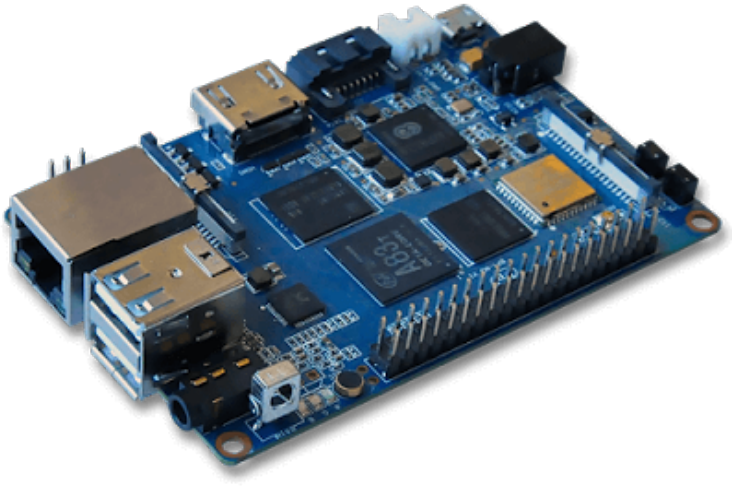
2.3.4 Banana Pi

Banana Pi je odnož projektu Raspberry Pi, která používa odlišné komponenty, ale s důrazem na zpětnou kompatibilitu tak, jak je to možné. [9]

Zatím poslední model nese označení Banana Pi M3. Tento model je vybaven 1.8GHz, 8-jádrovým procesorem Cortex A7 a 2GB DDR3 operační paměti. Toto zařízení, narozdíl od počítače Raspberry Pi, obsahuje větší množství vybavení již integrovaného přímo na desce. Model Banana Pi M3 je tedy, mimo jiné, již v základu vybaven například mikrofonem, wifi modulem s anténou, SATA konektorem či tlačítky reset a power.

Dále, obdobně jako počítač Raspberry Pi, obsahuje zařízení také 2 USB konektory, HDMI konektor, ethernet konektor, slot pro microSD kartu a sadu 40ti GPIO pinů. [10]

Zařízení také podporuje většinu operačních systémů jako počítač Raspberry Pi, včetně systému Raspbian. Pro zařízení Banana Pi také vznikl speciální systém s názvem Bananian, který je založený na systému Debian a je optimalizovaný pro počítače Banana Pi.



Obrázek 2.6: Počítač Banana Pi M3 [10]

Kapitola 3

Analýza a návrh

V této kapitole se zaměřím na analýzu a návrh řešení. Nejprve popíši použitý počítač Raspberry Pi a volbu operačního systému. Dále se budu věnovat analýze systému a jeho komponent. Nakonec se budu věnovat popisu uživatelských skriptů a bezpečnosti zařízení.

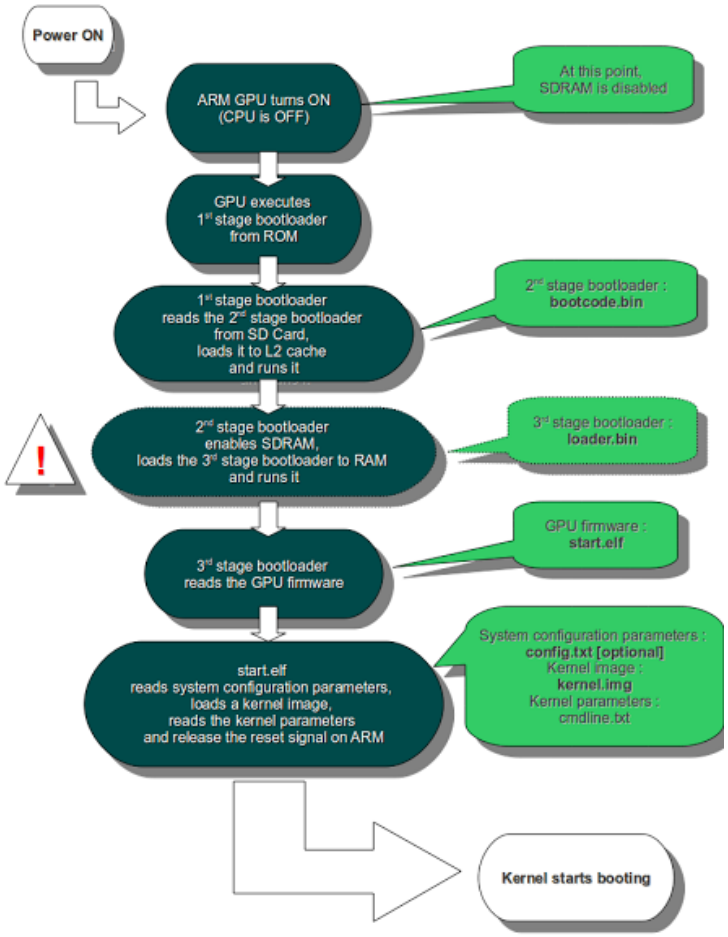
3.1 Počítač Raspberry Pi

Počítač Raspberry Pi byl vyvinut v roce 2012 britskou nadací Raspberry Pi Foundantion s cílem podpořit výuku informatiky na školách. První verze zařízení měla jednojádrový 700MHz procesor ARM11 a 256MB operační paměti. Model také obsahoval sadu GPIO pinů, USB a HDMI rozhraní a slot pro SD kartu. [11]

V této práci byla použita verze Raspberry Pi B+, která je vybavená jednojádrovým 700MHz procesorem řady ARM11 a 512MB operační paměti. Model je také vybaven 40 GPIO piny, 4 USB porty a na rozdíl od předchozích verzí obsahuje slot pro micro SD kartu. Počítač je obvykle dodáván pouze jako kus haradware bez instalovaného operačního systému. Systém je tedy nutné doinstalovat. Existuje celá řada systémů, které jsou určené přímo pro počítač Raspberry Pi. Některé z nich jsou dostupné na stránkách projektu (`https://www.raspberrypi.org/`). Dostupné systémy jsou obvykle dodávány jako již hotový image, určený k umístění na SD kartu, odkud počítač Raspberry Pi operační systém spouští.

Proces spuštění systému je na počítači Rapberry Pi poměrně specifický. Firmware zařízení vyžaduje, aby na SD kartě existoval oddíl naformátovaný se souborovým systémem typu FAT. Dále očekává, že se zde, mimo jiné soubory, nachází soubor kernel.img, což je jádro operačního systému, které je spuštěno v poslední fázi bootovací sekvence. Celý proces spouštění je znázorněn na obrázku 3.1. Na stejném odílu se také nachází soubor cmdline.txt, který obsahuje parametry předané jádru operačního systému při jeho spouštění (viz poslední krok na obrázku 3.1). Podíváme-li se na obsah souboru cmdline.txt na obrazu

systému Raspbian, který byl použit i pro toto zařízení (viz sekce 3.2), můžeme si všimnout parametru `root=PARTUUID=84fa8189-02`. Tento parametr říká jádru, jaké zařízení má být použito jako kořenový (root) filesystém během bootování. Zde tento parametr tedy udává, že root filesystém se nachází na druhém oddílu. Tento oddíl je již naformátován pomocí, pro Linuxové systémy typického, souborového systému ext4. Tuto skutečnost sděluje jádru parametr `rootfstype=ext4`. [12]



Obrázek 3.1: Proces spouštění počítače Raspberry Pi [13]

3.2 Volba OS

Pro zařízení bylo nutné zvolit vhodný operační systém. Zde jsem se rozhodl použít operační systém Raspbian, což je neoficiální port Debianu, určený právě pro zařízení Raspberry Pi. Operačních systémů, určených pro počítač Raspberry Pi (ARM architekturu), existuje celá řada. Většinou se jedná o port běžných Linuxových systémů, jako je Fedora, Ubuntu, Debian atd. Existuje i verze operačního systému Windows, která je rovněž určena pro zařízení s ARM architekturou. Nicméně většina těchto systémů je v aktuální verzi podporována pouze na zařízeních Raspberry Pi 2 a vyšších s architekturou alespoň ARMv7.

3.2.1 Raspbian

Raspbian je svobodný operační systém, založený na Debianu a je optimalizovaný právě pro platformu Raspberry Pi. První verze distribuce v roce 2012 nabízela na 35 000 balíčků z původního Debianu optimalizovaných pro Raspberry Pi. Nicméně Raspbian se stále vyvijí, a to s důrazem na zvýšení stability a výkonu mnoha dalších balíčků ze systému Debian. [14]

Raspbian vychází z armhf verze Debianu. Armhf znamená, že se jedná o Debian zkompilovaný pro ARM architekturu s takovým nastavením, které produkuje kód využívající takzvaný „hardware floating point“, „hard float“ ABI a tedy takový, který poběží na počítači Raspberry Pi. Hard float ABI je druh ABI (Application binary interface), které se od soft float ABI liší v práci s reálnými (float) čísly. Jde o to, jakým způsobem jsou předávána reálná čísla v parametru při volání funkcí nebo jakým způsobem je předávána návratová hodnota. Soft float ABI předává takové parametry pomocí celočíselných (integer) registrů, zatímco hard float ABI k tomuto účelu používá specializované vfp (floating point) registry. Tyto dvě ABI nejsou zcela kompatibilní, jelikož každé využívá jiné registry. [15]

Procesory řady ARM11, použité v modelu Raspberry Pi B+ využívají architekturu ARMv6 s „hard float“ ABI, z toho důvodu by se mohla armhf verze Debianu jevit jako vhodná volba pro použití na tomto zařízení. Nicméně tato verze není s použitým modelem počítače Raspberry Pi kompatibilní právě kvůli architektuře ARMv6 a tedy nebude fungovat. Přímo na wiki stránce projektu Debian je pro počítač Raspberry Pi s architekturou ARMv6 doporučeno využít systém Raspbian namísto armhf Debianu. [16]

3.3 Připojený displej

K zařízení bude dále připojen dotykový displej KeDei. Jedná se o 3.5"dotykový LCD displej, určený pro zařízení Raspberry Pi. Tento displej bude na zařízení sloužit k prezentaci výstupu z uživatelských skriptů. Displej se k počítači připojuje pomocí GPIO pinů přímo na desku.

3.4 Součásti

Funkcionalita celého zařízení je rozdělena na tři základní komponenty.

1. Komponenta pro správu zařízení
    - Komponenta spravuje uživatelské nastavení a vytváří webové rozhraní k jeho konfiguraci. Zároveň zpřístupňuje HTTP rozhraní pro přístup k webovým uživatelským skriptům.
2. Komponenta pro správu displeje
    - Tato komponenta obsluhuje připojený dotykový displej. Zobrazuje menu pro výběr jednotlivých skriptů a zajišťuje komunikaci s následující komponentou pro jejich spouštění.
3. Komponenta pro spouštění uživatelských skriptů
    - Tato komponenta je odpovědná za spouštění uživatelských skriptů. Čeká na požadavky z ostatních komponent a poté vyvolá příslušný skript. Výstup ze skriptu vrátí původnímu žadateli (displej/HTTP rozhraní).

Jednotlivé komponenty spolu vzájemně komunikují způsobem, který je popsán v sekci 3.8. Následující sekce obsahují detailnější popis všech komponent.

3.5 Komponenta pro správu zařízení

Jedním z požadavků na zařízení je možnost vzdálené konfigurace. Je tedy nutné zvolit vhodný způsob, jak uživateli zpřístupnit konfigurační rozhraní. Nejprve si tedy uveďme požadavky, které budou na toto rozhraní kladeny.

- Správa uživatelských skriptů
    - Umožňuje přidat, odebrat, zobrazit nebo změnit nastavení uživatelským skriptům.
- Správa certifikátu pro zabezpečené spojení
    - Certifikát bude použit k zabezpečení konfiguračního rozhraní i rozhraní pro přístup ke skriptům.
- Správa uživatelů pro spouštění webových skriptů
    - Zařízení bude umožňovat zabezpečení webového přístupu k uživatelským skriptům pomocí Basic autentizace. Za tímto účelem bude udržován seznam uživatelů, kteří mají přístup ke skriptům prostřednictvím webového rozhraní.

- Správa síťových připojení
  - Webové konfigurační rozhraní umožní upravit parametry webového serveru, jako je port pro HTTP i HTTPS a bezpečnostní certifikáty pro HTTPS.

Z výše popsaného plyne, že tato komponenta bude udržovat řadu informací (seznam skriptů, uživatelů a další nastavení). Seznam skriptů bude sdílen s ostatními komponentami. Přístup k rozhraní bude navíc zabezpečen pomocí digitálního certifikátu. Je tedy nutné zvolit způsob realizace komponenty s ohledem na popsané požadavky.

Nabízely se v zásadě dva způsoby řešení. Jedním z nich bylo vytvořit separátní aplikaci, která bude spuštěna na uživatelském počítači a pomocí zabezpečeného spojení bude komunikovat se zařízením Raspberry Pi. Druhou možností bylo vytvořit konfigurační rozhraní jako webovou aplikaci, ke které bude uživatel (správce) přistupovat pomocí webového prohlížeče. Zde jsem se rozhodl pro druhou variantu, jelikož je podstatně jednodušší.

3.5.1 Volba technologie

Pro webovou aplikaci bylo nutné zvolit vhodný programovací jazyk. Zde jsem mohl volit z řady variant. Při výběru jsem uvažoval nástroje, které umožňují přímo vytvořit webové rozhraní bez nutnosti instalace a konfigurace webového serveru. Z tohoto důvodu jsem zavrhl řešení za pomoci jazyka PHP, jelikož takové řešení typicky zahrnuje konfiguraci webového a PHP serveru. Volil jsem tedy mezi skriptovacími jazyky jako je Perl, Python, Ruby a Node.js. Volba nakonec padla na použití nástroje Node.js, se kterým jsem měl určité zkušenosti.

3.5.1.1 Node.js

Node.js je vývojová platforma pro jazyk JavaScript, která je určena pro vývoj serverových aplikací a je založená na Googlovském V8 JavaScript engine. Přesto, že Node.js funguje ve většině aplikací jako server, pracuje pouze nad jedním vláknem. Node.js aplikace jsou totiž řízené událostmi a fungují asynchronně. To znamená, že se zde způsob práce liší od obvyklého modelu, který zpracovává požadavky synchronně. Node.js je namísto toho pouze registruje a nečeká na jejich vyřízení. [17] Těmito požadavky může být třeba požadavek na čtení lokálního souboru či požadavek na přístup do databáze. Standardní synchronní model by s těmito požadavky pracoval sériově, tedy například by čekal (blokovaný), dokud by operační systém neotevřel požadovaný soubor či v druhém případě dokud by nepřišla odpověď z databáze. To tedy znamená, že pro plynulý provoz pracuje takový server obvykle ve více vláknech pro obsluhu jednotlivých klientů.

Node.js ale podporuje takzvané neblokující I/O, to znamená, že při vyřizování požadavků nedojde k zablokování procesu, který namísto toho pokračuje v další činnosti. Toto samozřejmě klade určité nároky na způsob práce programátora. Konkrétně to znamená, že programátor v situaci, kdy potřebuje například získat data z databáze, pouze zaregistruje zpětné volání (callback), které se vykoná až v situaci, kdy jsou data dostupná. Server mezitím pracuje dál, tedy může například obsloužit dalšího klienta. Takové chování je možné díky způsobu, jakým Node.js pracuje. Node.js pracuje nad takzvanou smyčkou událostí (event loop), během které pravidelně kontroluje, zda nedošlo ke konkrétním událostem a pokud ano, tak spustí na ně navázané callbacky. [18] Výhodou tohoto nástroje je právě to, že programátor pracuje pouze s jedním vláknem. Díky tomu odpadá starost o jejich synchronizaci a zjednodušuje se návrh aplikace.

3.5.2 Správa rozhraní pro přístup k webovým skriptům

Komponenta bude kromě správy uživatelských informací plnit ještě jeden úkol. Bude spravovat webové rozhraní pro přístup k webovým uživatelským skriptům. Toto se poměrně liší od původního účelu komponenty. Vzhledem k tomu by se dalo uvažovat nad tím, zda by tento úkol neměl být svěřen do správy nějaké jiné komponentě. Respektive zda by k tomuto účelu neměla nějaká nová vzniknout. Zde záleží na jaké úrovni chceme držet granularitu systému, tedy na tom, jaké množství různorodých úkolů by měly jednotlivé součásti zastupovat. V tomto případě ale musíme vzít v potaz i jeden technický detail. Dejme tomu, že by za účelem správy rozhraní pro přístup k webovým skriptům vznikla nová komponenta. Tato komponenta by samozřejmě musela vytvářet server, který by naslouchal na vnějším rozhraní zařízení (wifi/ethernet). Zde by ovšem mohlo dojít ke konfliktu. Původní komponenta pro správu uživatelského nastavení, tak jak je navržena, bude pracovat stejným způsobem a tedy by nebylo z technických důvodů možné, aby oba servery naslouchaly na stejném síťovém rozhraní a zároveň na stejném portu. Bylo by možné prohlásit, že každá aplikace bude muset běžet na jiném portu, nicméně takovému omezení bych se chtěl vyhnul.

3.5.3 Shrnutí

Vznikne tedy služba, jejímž účelem bude jednak vytvářet webové uživatelské rozhraní a dále spravovat databázi údajů ohledně uživatelských skriptů a dalšího nastavení. Tato služba bude dále v textu označována také jako Správce webového rozhraní.

Seznam uživatelských skriptů a s nimi souvisejících údajů jsou stěžejní informace, které jsou podstatné pro fungování celého systému. Tyto informace budou využívat i ostatní komponenty. Z toho důvodu bude nutné, aby je služba byla schopna v rámci systému sdílet. K tomuto účelu bude sloužit komunikační

middleware, který je blíže popsán v sekci 3.8, kde se věnuji i popisu fungování komunikace mezi službami.

3.6 Komponenta pro správu displeje

K zařízení bude připojený dotykový displej, který bude sloužit ke spouštění a prezentaci uživatelských skriptů. Je tedy nutné vytvořit aplikaci, která bude displej obsluhovat. Tato aplikace bude vytvořena pomocí programovacího jazyka C++ a knihovny Qt, což je multiplatformní framework, určený, mimo jiné, k tvorbě grafických uživatelských rozhraní.

3.6.1 Volba technologie

Volba programovacího jazyka C++ plyne již ze zadání práce. Samozřejmě existuje více možných jazyků, s nimiž by bylo možné grafické rozhraní vytvořit. Nicméně vzhledem k hardwarovým parametrům zařízení, pro které je aplikace určena, je použití programovacího jazyka C++ ideální volbou. Důvodem pro volbu tohoto jazyka je i samotné použití knihovny Qt. Samozřejmě existují portace této knihovny i pro řadu jiných programovacích jazyků, jako například pro jazyk Python, kde je její využití poměrně běžné. Nicméně samotná Qt knihovna je napsaná v jazyce C++ a pro něj je také primárně určena, vzhledem k tomu je tento jazyk nejvíce podporován. [19]

Jak je popsáno výše, pro grafické uživatelské rozhraní bude použita knihovna Qt. Qt je framework určený nejen pro vývoj grafických uživatelských rozhraní. Qt vyvíjí společnost Nokia pod vlastní licencí, ale pro nekomerční vývoj a pro studium je Qt nabízena pod svobodnou licencí. Projekty vytvořené s pomocí farmeworku Qt není možné přímo sestavit pomocí klasického C++ kompilátoru, jelikož Qt přináší určitá rozšíření jazyka C++ (např. systém signálu a slotu). Z toho důvodu je nutné zdrojové soubory nejprve před kompilací přeložit pomocí programu `moc` (Meta Object Compiler). Qt projekty také často obsahují UI soubory (přípona .ui), což jsou XML soubory, které vytváří nástroj QtCreator a které obsahují popis navrženého widgetu. Tyto soubory musí být také přeloženy na C++ kód, k tomu zde slouží program `uic` (User Interface Compiler). [20] Pro automatizaci těchto operací přichází Qt s nástrojem `qmake`, který slouží speciálně k sestavování Qt projektů (generuje soubor Makefile).

3.6.2 Popis fungování aplikace

Aplikace bude přímo odpovědná za prezentaci dat na připojeném dotykovém displeji a za interakci s uživatelem. Ve výchozím stavu nabídne aplikace na displeji seznam všech uživatelských skriptů. Aplikace komunikuje jednak s komponentou pro správu zařízení, odkud si načítá svou konfiguraci (seznam skriptů pro hlavní nabídku) a dále s komponentou pro správu skriptů, kterou

kontaktuje vždy s požadavkem na spuštění konkrétního skriptu. Způsob komunikace mezi komponentami je popsán v sekci 3.8. Aplikace umožní uživateli nastavit skript, který má být vykonán. Následně na displeji zobrazí výstup z tohoto skriptu.

Smysl tohoto způsobu prezentace dat se mírně liší od prezentace dat prostřednictvím webového rozhraní popsaného v sekci 3.5. Zobrazí-li si uživatel data prezentovaná nějakým webovým skriptem, tak ho zpravidla zajímá jejich aktuální podoba. Tedy výstup skriptu v okamžiku jeho spuštění. V případě dat prezentovaných prostřednictvím displeje je to ale jinak. Prezentace zobrazená na displeji zde zůstává do doby, než uživatel zvolí jinou. Vzhledem k účelu tohoto zařízení se předpokládá, že jedna prezentace může na displeji zůstat delší dobu. Z tohoto důvodu je nutný mechanizmus, který zajistí, aby prezentovaná data byla aktuální. Za tímto účelem bude aplikace pravidelně, dle konfigurace ve webovém rozhraní, opakovat požadavek na provedení aktuálního skriptu. Webové rozhraní tedy umožní ke každému skriptu tohoto druhu nastavit interval, v jakém se má aktualizovat. Pro skripty, které toto nevyžadují, bude možné pravidelnou aktualizaci deaktivovat.

3.6.3 Prezentace dat

Jak je popsáno výše, aplikace bude na displeji zobrazovat výstup z uživatelského skriptu. Ovšem kdyby byl tento výstup zobrazen jako prostý text tak, jak byl předán skriptem, byla by výsledná prezentace poměrně nepřehledná. Z toho důvodu je nutné najít způsob, jak uživateli umožnit prezentovaná data nějakým způsobem vhodně formátovat. K tomuto účelu jsem se rozhodl využít funkcionalitu knihovny Qt, která u vybraných widgetů, určených k zobrazování textu, umožňuje obohatit vstupní text o některé HTML tagy. Jedná se o podmnožinu HTML 4 a CSS stylů. Přesný přehled podporovaných tagů a CSS stylů je uveden na stránkách projektu (`http://doc.qt.io/qt-5.8/richtext-html-subset.html`). Takto tedy může uživatel pro vylepšení následné prezentace předávat výstupní data ze skriptu jako HTML dokument. Způsob tvorby uživatelských skriptů je blíže popsán v sekci 3.9.

3.6.4 Interakce uživatele se skriptem

Smyslem zařízení v tomto kontextu je umožnit uživateli zpracovat data dostupná (nejen) na internetu a dle vlastních potřeb je zobrazit na displej. Zde se tedy předpokládá statická prezentace, která dále neumožňuje uživateli s výstupem na displeji nějakým způsobem interagovat. Nicméně přesto jsem se rozhodl určitý způsob interakce se skriptem, prostřednictvím dotykového displeje, umožnit.

K tomuto účelu bude možné využít HTML značky. V seznamu podporovaných HTML tagů, který se nachází na adrese uvedené výše, ovšem nejsou uvedené žádné formulářové prvky (vstupní pole, tlačítka, atd.). Jedinou mož-

ností zde tedy je využít podporovaný HTML tag <a>, který běžně slouží k vložení odkazu. V tomto případě bude tag <a> fungovat jako tlačítko, při jehož stisku dojde k opětovnému vyvolání skriptu a hodnota atributu href tagu bude předána volanému skriptu. Způsob interakce skriptu tímto způsobem je blíže popsán v sekci 3.9.2.

3.6.5 Shrnutí

Pro obsluhu připojeného dotykového displeje vznikne aplikace, která bude na displeji zpřístupňovat nabídku pro volbu uživatelských skriptů a dále bude zajišťovat jejich spuštění prostřednictvím komunikiace se službou zajišťující správu skriptů (blíže popsanou v sekci 3.7). Aplikace bude dále v textu označována jako Správce displeje. Vytvořena bude pomocí programovacího jazyka C++ a knihovny Qt. Použité skripty mohou svůj výstup obohatit o základní HTML značky k vylepšení výsledné prezentace. Bude podporována omezená forma interakce uživatele se skriptem použitím HTML tagu <a>.

3.7 Komponenta pro spouštění uživatelských skriptů

Poslední součástí je služba, zajišťující spouštění uživatelských skriptů a předávání jejich výstupu. Jednou z možností, jak spouštět uživatelské skripty, by mohlo být jejich spouštění přímo ze služby obsluhující webové rozhraní nebo z aplikace pro obsluhu displeje. V takovém případě by příslušná služba spustila nový proces, který by vykonal konkrétní skript, předal jeho výstup a ukončil se. Předání výstupu z uživatelského skriptu by v tomto případě mohlo probíhat například pomocí roury, která by standardní výstup ze skriptu předala zpět službě, která proces spustila.

Výše popsané řešení má ovšem několik nedostatků. Jedním z nich je nutnost implementace spouštění a správy skriptů v obou službách (správce displeje a správce webového rozhraní). Dalším nedostatkem je vyšší výpočetní náročnost takového řešení. S každým požadavkem na provedení skriptu by v takovém případě bylo nutné opakovaně vytvářet nový proces, načíst a spustit požadovaný skript a nakonec předat jeho výstup prostřednictvím roury. Takové řešení dle měření způsobuje poměrně nezanedbatelnou zátěž procesoru, která během vykonávání skriptu dosahuje na použitém zařízení až k 40%. Největší podíl na této zátěži má právě spouštění samotného python procesu, samotné vykonání obvyklého uživatelského skriptu zabere minimum času (nepočítáme-li čas, kdy skript čeká na odpověď z internetu).

V tomto případě lze namítnout, že vzhledem k předpokládanému využití zařízení (domácí užití) není předpokládána jeho větší zátěž a tedy není nutné se takovíý problémem zabývat. Nicméně vzhledem k tomu, že zařízení kromě displeje umožňuje prezentovat data i prostřednictvím HTTP rozhraní, může

docházet k více paralelním požadavkům na provedení skriptu. Tyto požadavky by samozřejmě byly na straně služby, která je obsluhuje, serializované, ale zde by se již větší doba zpracování jednotlivých skriptů mohla projevit. Proto jsem se rozhodl využít o něco sofistikovanější řešení, které přináší i další výhody.

Navržené řešení předpokládá vznik další služby, která bude odpovědná za spouštění uživatelských skriptů. Tato služba bude dále v textu nazývána jako Správce skriptů. Správce skriptu bude k dispozici ostatním komponentám a bude vyřizovat jejich požadavky o provedení uživatelských skriptů. Odpovědí na požadavek bude výstup volaného skriptu.

Výhoda tohoto řešení je, že implementace spouštění skriptů je provedena na jednom místě a je oddělena od ostatních komponent. Díky tomu je například možné v budoucnu snáz rozšířit funkcionalitu zařízení například o zpracování jiného druhu skriptů (jiný skriptovací jazyk), či o zcela jiný způsob zpracování dat bez většího zásahu do ostatních komponent. Další výhoda tohoto návrhu oproti původně popsanému řešení spočívá ve vzniku nového komunikačního kanálu mezi správcem skriptů a službou žádající vykonání skriptu. Díky tomu je možné standardní i standardní chybový výstup využít k účelu logování průběhu skriptu. V původním návrhu byl standardní výstup využit k předání dat zpracovaných skriptem přes rouru do rodičovského procesu.

3.7.1 Volba technologie

Vzhledem k tomu, že uživatelské skripty budou psány v programovacím jazyce Python (viz sekce 3.9), bude i tato komponenta vytvořena pomocí stejného jazyka. Díky tomu bude možno uživatelské skripty spouštět přímo bez vytváření nového procesu, jelikož skripty budou za běhu importovány jako moduly. To zároveň umožní snazší interakci se skriptem, který bude výsledná data předávat jednoduše jako návratovou hodnotu z funkce (blíže v sekci 3.9).

3.7.2 Popis funkčnosti

Správce skriptů bude fungovat následujícím způsobem. Ihned po startu si služba vyžádá seznam všech skriptů a tyto skripty okamžitě naimportuje voláním funkce `__import__`. Služba poté přejde do výchozího stavu, kdy očekává, že mohou nastat dvě možné události. Jednou z nich je příchozí požadavek na spuštění skriptu. V takové situaci služba zavolá funkci start skriptu a její návratovou hodnotu použije jako odpověď na původní požadavek. Poté služba opět přejde do výchozího stavu. Druhou událostí, která může nastat je, že dojde k aktualizaci seznamu skriptů. V takovém případě služba odstraní všechny načtené skripty a opět naimportuje nové dle nového seznamu. Následně opět přechází do výchozího stavu.

Služba bude zaznamenávat průběh skriptu do logovacího souboru příslušného vykonávanému skriptu. Zde bude zaznamenáno vše co bude skript tisk-

nout na standardní i standardní chybový výstup. Log příslušející každému skriptu bude možné zobrazit ve webovém rozhraní zařízení.

3.7.3 Shrnutí

Vznikne nová služba, jejímž úkolem bude obsluha požadavků o vykonání skriptu. Služba bude vytvořena, stejně jako samotné skripty, pomocí jazyka Python. Výstup jednotlivých skriptů bude zaznamenáván do příslušných logovacích souborů, které budou dostupné přes webové rozhraní.

3.8 Komunikační vrstva

Jak bylo popsáno výše, existují celkem tři základní komponenty systému, jejichž správné fungování vyžaduje vzájemnou komunikaci. Bylo tedy nutné zvolit vhodný nástroj, který by tuto komunikaci umožňoval. Jelikož bude každá komponenta vytvořena pomocí jiného programovacího jazyka, bylo nutné zvolit nástroj, který bude podporovaný ve všech těchto jazycích. Další požadovanou vlastností je, aby zvolené řešení pokud možno co nejméně zatěžovalo výpočetní prostředky zařízení. Zde jsem se tedy poohlížel po nějakém nástroji, určeném právě k zprostředkování komunikace mezi procesy, který není závislý na konkrétním programovacím jazyce.

Nabízely se v zásadě dva způsoby řešení. První z nich je principiálně velmi jednoduchý. Spočívá v přímém propojení všech tří komponent navzájem. Komunikace by se tedy vždy řešila mezi dvěma konkrétnimi komponentamy. K předávání zpráv by mohl sloužit například HTTP protokol (REST API), který je vhodný pro přijímání požadavků a odesílání odpovědí. Takové řešení je sice principielně jednoduché a realizovatelné ve všech komponentách, nezávisle na faktu, že každá komponenta bude vytvořena pomocí jiného programovacího jazyka, ale co se implementace týče, tak je toto řešení poměrně nepraktické. Druhé řešení je oproti tomuto daleko robustnější.

Druhé řešení spočívá ve využití nějakého centralizovaného nástroje pro předávání zpráv. K tomuto účelu existuje řada vhodných řešení. Takovému nástroji se obecně říká message broker a slouží k předávání zpráv mezi příjemcem a odesílatelem pomocí jasně definovaného protokolu. Message broker řeší obvykle problém rozshálých systémů, kde je nutné zajistit, aby spolu všechny komponenty mohly komunikovat, ale vzhledem k velikosti systému je velmi nepraktické propojit všechny komponenty napřímo. Toto samozřejmě není, vzhledem k malému počtu komponent, problém této práce, nicméně uvedené nástroje řeší často i jiný problém. Tím je zajištění samotné komunikace napříč různorodým systémem a tedy odstínění programátora od návrhu a implementace komunikačního protokolu v každé komponentě. Takto tedy stačí již jen implementovat rozhraní dodané knihovny/modulu, který již zajistí zbytek. Message broker tvoří centrální bod komunikace celého systému. Běží tedy většinou jako separátní služba, která směruje zprávy mezi jednotlivými uzly.

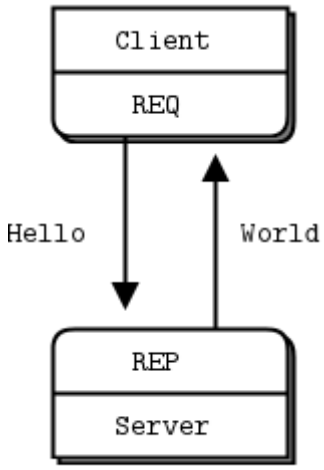
[21] Mezi zástupce message brokerů patří například RabbitMQ, Kafka, ActiveMQ, Kestrel a další. Pro všechny tyto nástroje, jak již bylo řečeno, platí, že fungují jako služba, která tvoří centrální uzel systému a předává zprávy mezi ostatními uzly. Nicméně toto, vzhledem k počtu komponent v systému, pro nás není nezbytně nutné. Naopak by zde bylo vhodné co nejjednoduší řešení, které by ale splňovalo všechny požadavky. Takovým řešením se zdá být nástroj ZeroMQ, který, na rozdíl od výše jmenovaných, funguje distribuovaně a tedy nutně nevyžaduje existenci jakéhokoliv centrálního směrujícího uzlu. Proto jsem se nakonec rozhodl právě k řešení s nástrojem ZeroMQ.

3.8.1 ZeroMQ

Ačkoli je v předchozím odstavci knihovna ZeroMQ uvedena v kontextu jiných systémů typu message broker, tak tato jím ve skutečnosti není. Jedná se o distribuovaný systém, kde jsou uzly propojeny přímo, což je tedy v rozporu s naší, výše popsanou, definicí message brokeru. ZeroMQ je pouze modul dostupný pro řadu programovacích jazyků, který zajišťuje přímé spojení mezi dvěma nebo více uzly pomocí síťového socketu, či jiným způsobem. ZeroMQ také zajišťuje udržování otevřeného spojení či jeho obnovení například po výpadku některého z uzlů. Dále také definuje a implementuje konkrétní komunikační protokol. Programátor tedy pouze prostřednictvím rozhraní ZeroMQ knihovny vkládá zprávy do otevřeného socketu a na druhé straně je opět čte. Způsob předávání zprávy (komunikační protokol) kompletně zajišťuje knihovna ZeroMQ. Pokud by bylo třeba, je možné zajistit obdobné fungování jako u klasického message brokeru. Stačí vytvořit zvláštní uzel, který bude sloužit ke směrování zpráv mezi ostatními. K tomu ZeroMQ nabízí speciální typ socketu s názvem `ROUTER`, který tvorbu takového uzlu umožňuje. V této práci si nicméně vystačíme, vzhledem k celkovému počtu komponent, s jejich přímým propojením. [22]

ZeroMQ podporuje řadu metod pro předávání zpráv. Jedná se o různé druhy socketů, které je možné vytvořit a kde každý slouží jinému účelu. Zde budou použity dva z nich. První je klasická metoda komunikace typu Request-Reply. Jedná se o nejjednodušší možný princip komunikace, který slouží k předání zprávy jednomu či více uzlům a k následnému přijetí odpovědi. Vzhledem k tomu, že se jedná o distribuovaný systém, tedy systém bez centrálního prvku, jsou oba dva komunikující uzly spojeny přímo a tedy musí v tomto případě jeden z nich fungovat jako server a druhý jako klient. Serverem je v tomto případě uzel, který slouží jako odpovídač (Reply - vytváří socket typu REP). Klientem je druhý uzel, který vytváří socket typu REQ. Princip komunikace je znázorněn na obrázku 3.2. Klient, tedy uzel který chce vytvořit požadavek (REQ), se připojí k serveru (REP) a odešle zprávu. Druhý uzel zprávu přijme a vrátí zpět odpověď. Přesto, že je možné k jednomu serverovému (REP) uzlu připojit více klientů, kde každý vytváří socket typu REQ, tak na straně serveru bude stále pouze jeden REP socket. Zde programátor při implementaci

logiky vyřizující odpovědi zpět žádajícímu uzlu, nemusí myslet na to, kam odpověď posílá. Knihovna ZeroMQ sama zajistí, že odpověď dorazí původnímu žadateli.

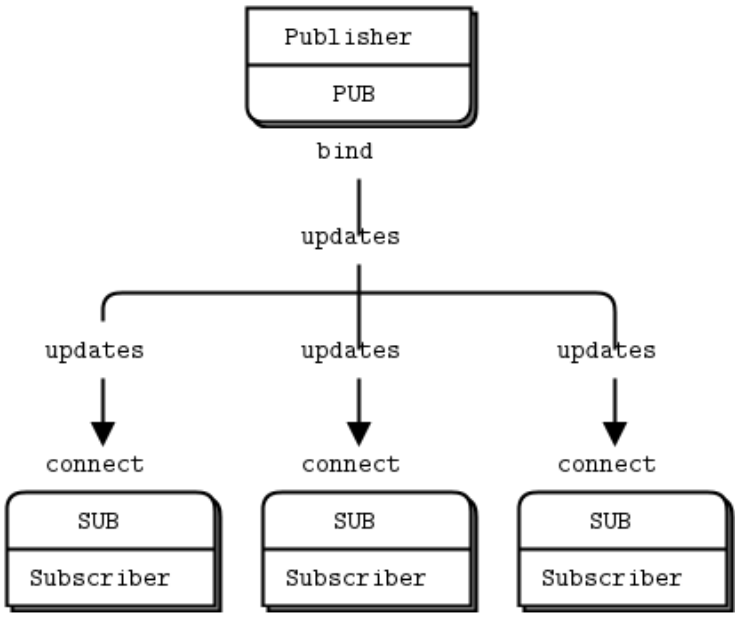


Obrázek 3.2: ZeroMQ, Request/Response komunikace [22]

V našem případě bude výše popsaná metoda komunikace použita ke dvěma účelům. Nejprve se tímto způsobem ihned po spuštění provede inicializace seznamu skriptů. Tento seznam udržuje služba Správce webového rozhraní. A dále bude využita vždy při vyvolání požadavku na spuštění skriptu službou Správce skriptů.

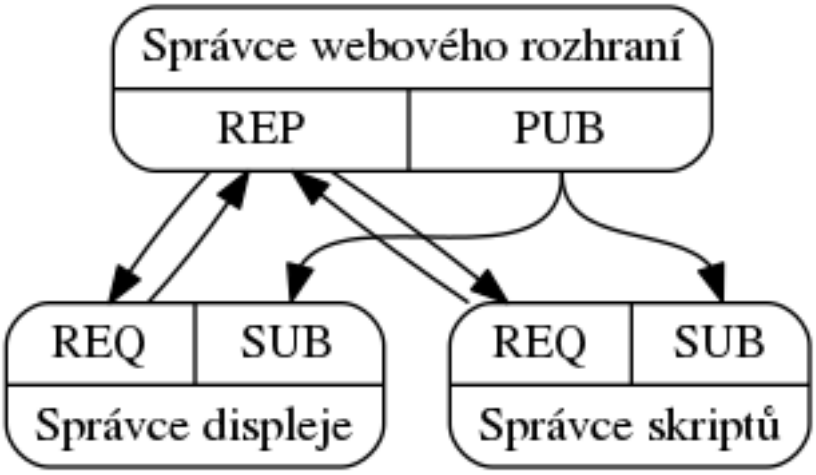
Druhý typ ZeroMQ socketů, který bude v systému využit, bude dvojice socketů PUB-SUB (publish - subscribe). Tyto typy socketů slouží k takovému druhu komunikace, kdy se jeden či více uzlů registruje (socket typu SUB) na příjem událostí generovaných jiným uzlem (socket typu PUB). Zde je opět nutné určit, který z této skupiny uzlů bude představovat server. Ostatní uzly budou představovat klienta. Je zřejmé, že serverovým uzlem musí být ten uzel, který generuje jednotlivé události. Tento uzel vytvoří opět pouze jeden socket typu PUB a ostatní se k němu připojí prostřednictvím SUB socketu. Serverem generované události jsou v tomto případě zprávy, které se skládají ze dvou částí. První část nese název generované události a druhá její obsah (zprávu, kterou se server snaží předat registrovaným klientům). V tomto případě je komunikace jednostranná, tedy neprobíhá žádná odpověď na přijatou událost. Princip tohoto druhu komunikace je znázorněn na obrázku 3.3.

V našem případě bude výše popsaná metoda komunikace sloužit k aktualizaci seznamu skriptů v případě, že uživatel provede nějakou změnu v administračním rozhraní zařízení. V takové situaci dojde na straně Správce webového rozhraní k vygenerování události s názvem `scriptListUpdate` jejímž obsahem bude aktuální seznam skriptů, včetně příslušných údajů. Model



Obrázek 3.3: ZeroMQ, Publish/Subscribe komunikace [22]

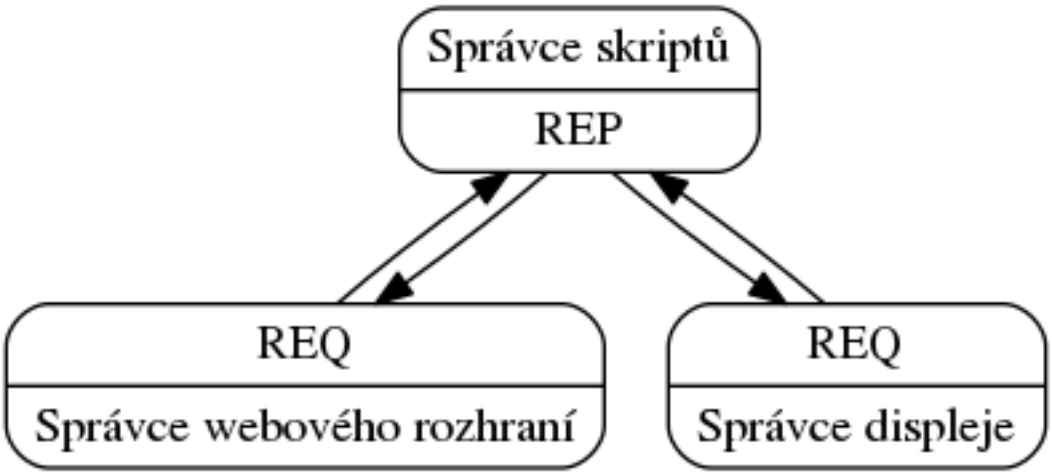
komunikace, zajišťující počáteční načtení uživatelských skriptů pomocí REQ-REP socketů a dodatečnou aktualizaci za běhu systému pomocí PUB-SUB socketů, je znázorněn na obrázku 3.4.



Obrázek 3.4: Mechanizmus načítání seznamu skriptů pomocí ZeroMQ

Obrázek 3.5 znázorňuje schéma komunikaci komponent s komponentou

Správce skriptů při požadavku na spuštění skriptu.



Obrázek 3.5: Mechanizmus spouštění skriptů pomocí ZeroMQ

3.9 Tvorba uživatelských skriptů

Účelem zařízení je prezentovat data tak, jak bude uživatel (správce) požadovat. K tomuto účelu budou, dle zadání, sloužit uživatelské skripty, které bude možné do zařízení prostřednictvím konfiguračního rozhraní nahrávat. Tyto skripty byly již zmíněny v předchozím textu. Nyní se tedy zaměřím na jejich detailnější popis.

Pro uživatelské skripty je nutné zvolit vhodný skriptovací jazyk. K tomuto účelu jsem zvolil jazyk Python, který je dosti rozšířený a dobře se pro psaní uživatelských skriptů hodí, jelikož v nich jde převážně o pohodlné zpracování textu. S tímto si samozřejmě poradí stejně obstojně i řada jiných skriptovacích jazyků. Jazyk Python byl mezi nimi vybrán i na základě osobních preferencí. Do budoucna je samozřejmě možné rozšířit funkcionalitu zařízení i o další jazyky.

Uživatelské skripty jsou rozděleny na dva druhy, jeden druh uživatelských skriptů slouží pro prezentaci dat prostřednictvím webového rozhraní zařízení a druhý pro prezentaci pomocí připojeného dotykového displeje. Pro oba druhy skriptů platí několik pravidel.

1. Skripty musí být vytvořeny pomocí skriptovacího jazyka Python.

2. Skripty musí obsahovat funkci s názvem start, která tvoří vstupní bod pro spuštění skriptu. Tato funkce se tedy volá vždy při vyvolání požadavku uživatelem.

3. Funkce start vždy přijímá jeden argument, jeho význam se liší podle druhu skriptu.

Následující sekce se zaměřují na popis pravidel a chování specifických pro každý druh skriptu.

3.9.1 Webové uživatelské skripty

Tyto skripty jsou vyvolány vždy v okamžiku, když uživatel provede příslušný HTTP požadavek na webové rozhraní zařízení. Požadavek musí být proveden pomocí HTTP metody GET na zdroj dle konfigurace. Systém poté spustí funkci start v uživatelském skriptu a předá jí seznam parametrů, které obsahoval původní HTTP požadavek v URL. Seznam parametrů je předán jako jediný parametr, který obsahuje strukturu slovníkového typu (dict). Klíče v této struktuře představují název parametru a hodnota odpovídá hodnotě parametru v původním HTTP požadavku.

Od funkce start se v tomto případě očekává, že vrátí novou strukturu, která bude použita jako HTTP odpověď uživateli. Struktura musí být opět slovník (dict) s následujícími předepsanými prvky:

```
{
        httpStatus: <int>,
        contentType: <str>,
        payload: <str>
}
```

- Hodnota prvku **httpStatus** bude použita jako HTTP status v odpovědi
- Hodnota prvku **contentType** bude sloužit jako hlavička Content-Type v odpovědi
- Hodnota prvku **payload** představuje obsah těla odpovědi

3.9.2 Skripty pro připojený displej

Tyto skripty slouží k prezentaci dat na připojeném dotykovém displeji. Výstupem skriptu je vždy řetězec, který bude na displeji zobrazen. Pro vylepšení výsledné prezentace bude systém podporovat podmnožinu HTML4 tagů a CSS kaskádových stylů, jimiž může být výstupní řetězec obohacen. Možnosti zpracování HTML a CSS zdrojů vychází z možností knihovny Qt, použité při tvorbě grafického rozhraní zařízení a jejího widgetu QTextBrowser, který slouží k zobrazení výstupu ze skriptu. Seznam podporovaných HTML tagů a kaskádových stylů je uvedena na adrese `doc.qt.io/qt-5.8/richtext-html-subset.html`.

Skripty jsou vyvolány ve třech situacích. První z nich je, když uživatel spustí skript v menu na dotykovém displeji a druhou, když dojde k pravidelné

aktualizaci, kterou je možné nastavit zvlášť pro každý skript v konfiguračním rozhraní. Třetí, poslední situace, kdy dochází k vyvolání skriptu je, když dojde k interakci s uživatelem pomocí stisku HTML odkazu. Jedním z podporovaných HTML tagů je totiž i tag <a> pro vytvoření linku. Tento tag je zde využit právě k interakci uživatele se spuštěným skriptem. Pokud totiž dojde ke stisknutí tohoto odkazu, systém vyvolá skript znovu a jako parametr funkci start předá řetězec obsahující hodnotu atributu `href` z HTML tagu <a>. Systém předává funkci start parametr i v předchozích dvou situacích. Pokud se jedná o první situaci, kdy uživatel právě vybral skript z menu, je funkci start předán jako parametr prázdný řetězec. Pokud se jedná o pravidelnou aktualizaci dle uživatelského nastavení, je funkci start předán vždy stejný řetězec, se kterým byla volána naposledy. Toto je z důvodu, aby při pravidelné aktualizaci nedošlo ke změně prezentace, jelikož se zde vychází z předpokladu, že předávaný parametr může mít vliv na skriptem prezentovaná data. Příklad tvorby tohoto druhu skriptů znázorňuje následující ukázka.

```
def start(link):
    return """
        <html>
            <body>
                <h1>Ukazkovy skript</h1>
                <img src=file://tmp/test.png>
            </body>
        </html>
            """
```

Jak je vidět v ukázce výše, je podporován i tag <img>, který slouží pro vložení obrázku. Zde ovšem není možné použít klasický internetový zdroj, použitý widget by si s tím nedokázal poradit. Namísto toho je možné použít soubor, který se nachází lokálně v souborovém systému. Takový zdroj musí být uveden prefixem „file://“.

3.10 Zabezpečení

Jednou z otázek, které je nutné věnovat pozornost, je otázka zabezpečení práce se zařízením. Zařízení je určeno především pro domácí užití, tedy se nepředpokládá, že bude běžně odolávat jakémukoliv druhu útoku. Nicméně data, která budou zařízením prezentována mohou být citlivá, proto je nutné se touto otázkou zabývat.

V této části se zaměřím na problém zabezpečení ze dvou úhlů pohledu. Prvním je zabezpečení vnějšího rozhraní zařízení. Tedy způsob, jak ochránit komunikaci mezi uživatelem a webovým rozhraním zařízení a také jakým způsobem omezit přístup pouze oprávněným osobám. Dále se budu věnovat otázce vnitřního zabezpečení, tedy jakým způsobem bude zařízení chráněno před zneužitím prostřednictvím samotných uživatelských skriptů.

3.10.1 Webové rozhraní

Způsob zabezpečení webového rozhraní byl naznačen již v sekci 3.5, kde se píše, že bude umožněno nahrávání bezpečnostních certifikátů k zabezpečení komunikace. Tyto certifikáty budou sloužit k šifrování HTTPS komunikace jednak pro přístup k administračnímu rozhraní zařízení a jednak pro přístup k webovým skriptům. Aby bylo možné omezit přístup k webovým skriptům pouze pro oprávněné osoby bude možné pro jednotlivé skripty nastavit povinnost autentizace uživatele. Webové rozhraní bude v tomto případě podporovat jednoduchou (Basic) autentizaci. Při použití tohoto způsobu autentizace nedochází k žádnému šifrování uživatelského jména a hesla v HTTP dotazu. Z toho důvodu je vhodné tuto metodu kombinovat s přístupem přes zabezpečený protokol HTTPS.

3.10.2 Uživatelské skripty

Zabezpečení zařízení proti hrozbám ze strany uživatelských skriptů je komplikovaný problém a je otázka, zda je nutné se mu vůbec věnovat. Uživatelské skripty se do zařízení vkládají prostřednictvím zabezpečeného webového rozhraní, ke kterému je možné přistupovat pouze se znalostí účtu správce. Z toho důvodu lze prohlásit, že správce jako jediná osoba, která je schopna skript na zařízení umístit, je plně odpovědna za jeho obsah. Takové prohlášení je poměrně alibistické, nicméně zajistit naprosto spolehlivý mechanismus ochrany proti takovéto hrozbě je nemožný úkol. Z toho důvodu se v otázce tohoto druhu zabezpečení opírám především právě o výše psané prohlášení.

Kapitola 4

Realizace

V této kapitole se věnuji samotné realizaci projektu a případně problémům, které se při ní vyskytly. Nejprve se zaměřím na přípravu prostředí, tedy instalaci a konfiguraci operačního systému. Dále se věnuji popisu realizace všech tří komponent a nakonec samotné konfiguraci zařízení tak, aby celý systém fungoval dle očekávání.

4.1 Příprava prostředí

Jako operační systém byl zvolen systém Raspbian Jessie. Toto je poslední verze systému Raspbian, která byla v době realizace dostupná. K zařízení je připojený dotykový 3.5"displej KeDei, který je určený specielně pro počítač Raspberry Pi a je kompatibilní, mimo jiné, také s operačním systémem Raspbian. Displej se k počítači Raspberry Pi připojuje prostřednictvím GPIO pinů na základní desce. Pro správnou funkčnost je nutné do systému doinstalovat ovladač displeje, který je dostupný na stránkách výrobce (www.kedei.net). Po instalaci ovladače ovšem displej fungoval pouze částečně. Obraz byl v pořádku, ale z nějakého důvodu nereagoval displej na dotek. Proto jsem zkusil druhou možnost a to použít image operačního systému s již nainstalovaným ovladačem, který je rovněž dostupný na stránkách výrobce. Dostupný operační systém s předinstalovaným ovladačem je opět systém Raspbian Jessie. Tímto způsobem již displej fungoval bez problému.

Systém Raspbian je standardně dostupný ve dvou verzích - plnohodnotná verze a verze lite. Lite verze narozdíl od té plnohodnotné neobsahuje součást Pixel (**P**i **I**mproved **X**windows **E**nvironment, **L**ightweight), což je prostředí plochy používané v systému Raspbian. Jelikož aplikace pro obsluhu displeje neběží jako klasická okenní aplikace, není pro fungování systému prostředí plochy potřebné. Aplikace navíc přistupuje přímo na framebufer (viz sekce 4.3.1), není tedy nutné ani aby běžela služba X Server. Nicméně systém Raspbian, dostupný na stránkách výrobce displeje KeDei s již předinstalovaným ovladačem, je pouze v plnohodnotné verzi. Bylo tedy nutné systém nakonfigurovat

tak, aby se prostředí plochy po startu nespouštělo. K takovému nastavení slouží v systému Raspbian nástroj `raspi-config`, což je vlastně interaktivní shell skript, určený k základní konfiguraci systému.

4.2 Správce webového rozhraní

Správce webového rozhraní je komponenta, jejíž primární účel je zpřístupnit uživateli konfiguraci zařízení prostřednictvím webového rozhraní. Komponenta ale také obsluhuje přístup k webovému rozhraní uživatelských skriptů. Vzniklá aplikace je vytvořena pomocí programovacího jazyka Node.js. V této sekci se budu věnovat popisu vývoje aplikace, její činnosti a také popisu práce s uživatelským rozhraním.

4.2.1 Inicializace aplikace

Při startu aplikace je nejprve nutné provést její inicializaci. Tato inicializace se skládá z několika následujících kroků.

1. Start HTTP serveru
2. Start HTTPS serveru, je li server nakonfigurován
3. Služba vytváří ZeroMQ socket typu PUB. Tento socket slouží k rozesílání událostí o změně uživatelských skriptů. Ostatní služby se připojují pomocí SUB socketu
4. Služba vytváří ZeroMQ socket typu REP, pro příjem žádosti o seznam skriptů
5. Služba spouští příkaz `systemd-notify --redy`, čímž dává službě systemd najevo, že je plně zinicializována (viz sekce 4.5.1)

4.2.2 Vývoj aplikace

Vývoj aplikace je možné rozdělit do několika částí:

- Vývoj webového administračního rozhraní
- Vývoj webového rozhraní pro přístup ke skriptům
- Obsluha ZeroMQ socketů

V následujících sekcích se zaměřím na tyto jednotlivé součásti.

4.2.2.1 Webové konfigurační rozhraní

Tato součást aplikace je odpovědná za realizaci webového rozhraní, jehož prostřednictvím může uživatel spravovat seznam uživatelských skriptů a s tím souvisejícího nastavení.

Webové rozhraní je vytvořeno pomocí modulu express a ejs. Express je framework pro vývoj webových aplikací určený pro Node.js. Ejs je šablonovací nástroj, který je možné použít společně s modulem express a který umožňuje oddělit aplikační logiku webové aplikace od té prezentační. V praxi to funguje tím způsobem, že vytváříme ejs skripty (přípona .ejs) obsahující HTML kód uživatelského rozhraní a přímo do něj jsou vloženy sekvence kódu, který je vykonán na straně serveru a umožňuje dynamicky dotvářet podobu výsledného HTML dokumentu předaného klientovi jako odpověď na HTTP požadavek. Ejs skriptu je možno předat při spouštění data, nad kterými bude pracovat. Ejs skript je poté zapracuje do HTML šablony (odtud název šablonovací nástroj).

Aplikace má na starosti udržování celé řady údajů. Bylo tedy nutné zvolit vhodný způsob jak tyto údaje uchovat. Zde tedy připadalo v úvahu využít nějakou databázi. Vzhledem k tomu, na jakém zařízení celý systém běží, jsem hledal co nejméně náročné řešení. Zde se tedy zdálo vhodné využít nějakou NoSQL databázi. NoSQL databáze jsou nerelační databáze, které svá data neuchovávají v tabulkách a oproti klasickým relačním databázím jsou obecně, co se týče použití i nároků, poměrně jednodušší. Vzhledem k požadavkům systému na uložená data jsem se zde rozhodl využit databázi NeDB. NeDB je, co se udržování dat týče, velice jednoduché řešení. NeDB udržuje data jednak v paměti a také v obyčejném textovém souboru, kde jsou data uložena jako JSON dokument. [23] Toto řešení je pro požadovaný účel ideální, jelikož databáze bude sloužit pouze pro účely této aplikace a zároveň množství uložených dat nebude příliš velké (Pouze uložená konfigurace, maximálně 10 uživatelů a seznam skriptů). Výhodou tohoto řešení je, že nevyžaduje, aby v systému běžela další služba, která by se starala o správu dat. Celé řešení je čistě javascriptový modul pro Node.js.

Aplikace umožňuje spravovat připojení k síti wifi. K tomuto účelu je využit modul pi-utils, což je sada nástrojů pro vývoj na Raspberry Pi. Modul, kromě jiného, obsahuje také nástroje pro správu wifi připojení. K tomuto využívá modul nástroj `wpa_cli`, což je textový frontend pracující s nástrojem `wpa_supplicant`. `Wpa_supplicant` je demon běžící na pozadí systému, který spravuje připojení k bezdrátovým sítím. [24] `Wpa_supplicant` ukádá poslední připojenou síť, díky tomu je zajištěno, že po restartu se zařízení opět připojí ke stejné sítí, je li v dosahu.

4.2.2.2 Rozhraní pro přístup ke skriptům

Tato součást aplikace je odpovědná jednak za vytváření rozhraní pro přístup k webovým uživatelským skriptům a dále za komunikaci směrem ke Správci

skriptů. Při příchozím požadavku provede aplikace následující operace:

1. Aplikace ověří, že existuje požadovaný skript, daný cestou v URL požadavku

2. Aplikace ověří, zda protokol použitý při požadavku (HTTP nebo HTTPS) odpovídá protokolu nastavenému pro daný skript, případně klienta přesměruje

3. Aplikace ověří, zda skript vyžaduje autentizaci uživatele. Pokud ano, tak si od uživatele vyžádá jméno a heslo (Jedná se o jednoduchou Basic autentizaci)

4. Aplikace vytvoří požadavek na spuštění příslušného skriptu prostřednictvím zeromq modulu (blíže v sekci 4.2.2.3), odpověď vrátí jako odpověď na původní HTTP požadavek

4.2.2.3 Obsluha ZeroMQ socketů

Další činosti této komponenty je obsluha požadavků na seznam uživatelských skriptů, případně distribuce nového seznamu v případě jeho změny uživatelem. K tomuto slouží ZeroMQ sockety REP a PUB. Pokud si některá jiná komponenta vyžádá seznam skriptů, dojde k tomu prostřednictvím socketu typu REP. V případě, že dojde ke změně seznamu skriptů (uživatel provede změnu v administračním rozhraní) dojde k vygenerování události `scriptListUpdate`, která se rozšíří prostřednictvím PUB socketu mezi ostatní komponenty. V obou případech je obsahem zprávy nesoucí seznam skriptů stejný JSON dokument, který má následující strukturu.

```
{
    displayScripts: <Array>,
    webScripts:     <Array>
}
```

Struktura zprávy se skládá ze dvou polí, displayScripts a webScripts. Jak vyplývá z jejich názvů, pole displayScripts obsahuje seznam skriptů pro připojený displej a pole webScripts obsahuje seznam skriptů pro webové rozhraní. Struktura prvků pole displayScripts je následující:

```
{
    scriptName:  <String>,
    title:       <String>,
    description: <String>,
    reloadTime:  <Number>
}
```

- Atribut scriptName nese název skriptu tak, jak je uložen v souborovém systému
- Atribut title nese titulek, který bude zobrazen u jednotlivých skriptů ve výchozím menu na displeji
- Atribut description nese popis skriptu, který bude zobrazen opět v hlavním menu na displeji pod titulkem
- Atribut reloadTime nese čas pro pravidelnou aktualizaci skriptu

Všechny tyto atributy se nastavují v sekci Skripty ve webovém administračním rozhraní.

Struktura prvků pole webScripts je následující:

```
{
    scriptName:  <String>
}
```

Jak je vidět, tato struktura je velice jednoduchá. Obsahuje pouze jeden atribut, který nese název skriptu. Žádné další atributy zde nejsou nutné, toto pole slouží pouze komponentě Správce skriptů, který na jeho základě importuje jednotlivé skripty (viz sekce 4.4.1).

4.2.3 Práce s webovým rozhraním

Webové konfigurační rozhraní je rozděleno do následujících sekcí:

- Skripty
- Uživatelé
- Soubory
- Wifi
- Nastavení

Při přístupu k rozhraní je vyžadováno přihlášení účtem správce. Tento účet lze změnit v sekci nastavení. Výchozí údaje, tedy jméno a heslo, mají shodnou hodnotu: „admin“.

4.2.3.1 Skripty

V této sekci je možné přidávát, odebírat nebo modifikovat seznam uživatelských skriptů. Tato sekce je zobrazena na obrázku 4.1. Se skripty pro displej a skripty pro webové rozhraní se pracuje zvlášť, jelikož ke každému se nastavují jiné parametry. Skript pro displej mají následující seznam parametrů:

- **Název** - Unikátní název skriptu (bez přípony)
- **Titulek** - Titulek skriptu, který bude zobrazen v menu na displeji
- **Popis** - Krátký popis skriptu, který bude zobrazen v menu pod titulkem
- **Čas opakování** - Interval pro pravidelnou aktualizaci skriptu

Skripty pro webové rozhraní mají tyto parametry:

- **Název** - Unikátní název skriptu (bez přípony)
- **Cesta** - Poslední část URL cesty, kde bude skript dostupný. Webové skritpy jsou všechny dostupné na URL /webScripts/<skript>. Tedy bude-li parametr Cesta obsahovat například hodnotu „weather“, bude skript dostupný na URL cestě /webScripts/weather
- **Zabezpečení** - Volba protokolu HTTP či HTTPS
- **Autentizace** - Udává, zda má být pro daný skript vyžadována autentizace uživatele. Pokud ano, bude se uživatel ověřovat vůči seznamu v sekci Uživatelé

V této sekci je možné rovněž u jednotlivých skriptů zobrazit log soubor. Log soubor obsahuje vše, co příslušný skript během svého běhu tisknul na standardní i standardní chybový výstup.

4.2.3.2 Uživatelé

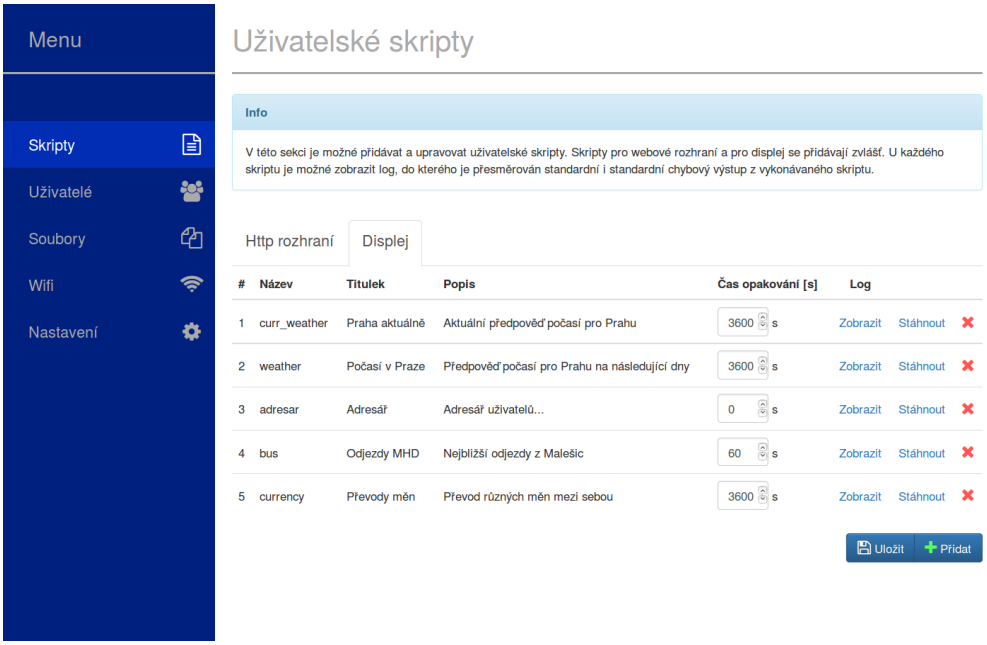
V této sekci je možné spravovat seznam uživatelů, který slouží pro připojení k webovým skriptům (pokud tyto vyžadují autentizaci uživatele). Je možné přidat maximálně 10 uživatelů. U každého uživatele je možné spravovat seznam skriptů, ke kterým má povolený přístup.

4.2.3.3 Soubory

V této sekci je možné přidávat dodatečné soubory, které můžou být následně využity v samotných skriptech (např. obrázky). Soubory jsou ukládány do adresáře `/opt/user_scripts/user_files`, zde s nimi mohou skripty libovolně nakládat.

4.2.3.4 Wifi

Sekce Wifi slouží pro připojení ke konkrétní wifi síti. V sekci je zobrazena tabulka s nabídkou dostupných bezdrátových sítí.



Obrázek 4.1: Ukázka webového konfiguračního rozhraní

4.2.3.5 Nastavení

Sekce Nastavení slouží jednak ke změně účtu správce a dále k úpravě základních parametrů webového serveru (HTTP a HTTPS porty, certifikáty).

4.3 Správce displeje

4.3.1 Příprava prostředí

Aplikace pro ovládání displeje byla vytvořena pomocí programovacího jazyka C++ a knihovny Qt. Zde jsem chtěl využít poslední verzi knihovny, kterou byla v době realizace verze s označením 5.8. Standardní instalace knihovny Qt probíhá pomocí online instalátoru, který je ke stažení na oficiálních stránkách projektu (`http://www.qt.io`). Tento instalátor po spuštění umožní stáhnout libovolnou verzi knihovny, či přímo zdrojový kód. Nicméně pro platformu Raspbery Pi nejsou binární verze knihovny touto cestou dostupné. Další možností bylo nainstalovat knihovnu z online repozitáře systému Raspbian jako balík. Zde ale byla v době realizace poslední dostupná verze 5.3.1. Proto jsem se rozhodl sestavit knihovnu ze zdrojových kódů přímo pro cílovou platformu Raspbery Pi. Sestavit knihovnu ze zdrojových kódů přímo na cílovém zařízení ovšem nebylo, vzhledem k velikosti knihovny, z praktických důvodů možné. Doba kompilace by na počítači Raspbery Pi byla až příliš dlouhá. Vzhledem

k tomu jsem se tedy rozhodl sestavit knihovnu na jiném zařízení (notebook HP s procesorem Intel core i5) pomocí křížového kompilátoru (cross compiler). Díky tomu bylo možné stejným způsobem sestavovat i výslednou aplikaci, což ušetřilo značný čas v průběhu vývoje oproti případné kompilaci přímo na cílovém zařízení.

Použitý křížový (cross) kompilátor byl arm-linux-gnueabihf, což je vlastně GNU C++ kompilátor pro armhf architekturu. Bylo nutné nastavit kompilátoru správný sysroot, což je parametr, který určuje, kde bude kompilátor hledat hlavičkové soubory a knihovny nutné pro kompilaci zdrojového kódu. Standardně gcc kompilátor hledá tyto soubory v adresářích vůči kořenovému (root) adresáři, tedy hlavičkové soubory hledá v adresáři /usr/include a knihovny v adresáři /usr/lib. Zde ale bylo nutné použit hlavičkové soubory a knihovny přímo z cílového zařízení Raspberry Pi. Jako parametr sysroot bylo tedy nutné nastavit adresář /mnt/rasp-pi-rootfs, kde byl připojený image souborového systému cílového zařízení. Nyní tedy kompilátor hledal hlavičkové soubory v adresáři /mnt/rasp-pi-rootfs/usr/include a knihovny v adresáři /mnt/rasp-pi-rootfs/usr/lib. Zde se ale vyskytl drobný problém. Soubory knihoven v adresáři /usr/lib, nejsou většinou přímo požadované binární soubory, ale pouze odkazy na ně. Nicméně tyto odkazy jsou tvořeny absolutní cestou, tedy vůči kořenovému adresáři, ale jelikož je souborový systém zařízení pouze připojen do adresáře /mnt/rasp-pi-rootfs, budou tyto absolutní cesty vztaženy oproti standardnímu kořenovému adresáři „/“, což samozřejmě způsobí, že budou neplatné. Zde se nabízejí dvě možnosti jak tento problém vyřešit. Jednou z nich by bylo dočasně změnit kořenový adresář na adresář /mnt/rasp-pi-rootfs, příkazem `chroot /mnt/rasp-pi-rootfs`. Tím by byly odkazy na knihovny opět platné (jelikož by nyní absolutní cesta směřovala vůči správnému adresáři). Toto řešení by ale vyžadovalo zároveň přesunout celý projekt do téhož adresáře, jelikož by jinak nebyl se změněným root adresářem přístupný. Dále by to samozřejmě znamenalo pracovat vždy v takto upraveném prostředí. Zde se jako jednoduší řešení tedy jevila druhá možnost, a to samotná úprava linků z absolutní cesty na relativní, čímž se docílí toho, že budou linky vždy platné při jakékoliv změně kořenového adresáře. Pro tuto úpravu stačilo vytvořit následující jednoduchý skript:

```
#!/bin/bash

DIR="$1/usr/lib/arm-linux-gnueabihf/"

find $DIR -maxdepth 1 -type l -exec file {} \; |\
sed -n 's/^\(.*\): .* \(\/.*\)$/\1 \2/gp'      |\
while read link target
do
    echo "repair: $link -> ../../..$target"
    rm $link
```

```
    ln -s "../../..$target" "$link"
done
```

Nyní již sestavení knihovny nic nebránilo. Zkompilovaná knihovna byla jednak umístěna na paměťovou kartu zařízení. Image této karty byl připojen také na počítači kde vývoj probíhal, jelikož to bylo nezbytné pro úspěšnou křížovou kompilaci aplikace.

Takto sestavenou aplikaci bylo nutné na zařízení Raspberry Pi spouštět s parametrem `-platform linuxfb`. Tím aplikaci říkáme, že má přistupovat přímo k framebufferu zařízení. [25] Aplikace tedy nepracuje s X serverem a ten tedy vůbec nemusí běžet.

4.3.2 Vývoj aplikace

Samotná aplikace funguje jako stavový automat. Jednotlivé stavy automatu odpovídají jednotlivým obrazovkám, které mohou být na displeji zobrazeny. Aplikace se může nacházet v následujících stavech:

- `STATE_INIT` (**Inicializace aplikace**) - stav ihned po spuštění systému, aplikace se snaží získat seznam skriptů prostřednictvím ZMQ knihovny. V tomto stavu je na obrazovce animace znázorňující, že je zařízení zaneprázdněno.
- `STATE_MENU` (**Výchozí stav**) - na obrazovce je zobrazeno menu s volbou jednotlivých skriptů. Tento stav je znázorněn na obrázku 4.2
- `STATE_LOADING` (**Načítání uživatelského skriptu**) - tento stav nastává v okamžiku, kdy uživatel vybere konkrétní skript z menu. Na obrazovce je v tomto stavu stejná animace, jako v případě inicializace aplikace. V tomto stavu probíhá komunikace se správcem skriptů prostřednictvím ZeroMQ knihovny. Poté, co komunikace skončí, přechází aplikace do stavu `STATE_SHOW_SCRIPT`
- `STATE_SHOW_SCRIPT` (**Zobrazení výstupu ze skriptu**) - v tomto stavu aplikace zobrazuje na displeji data, která jí předala služba pro správu skriptů.
- `STATE_SHUT_DOWN` (**Vypnutí systému**) - v tomto stavu se aplikace nachází, pokud uživatel na výchozí obrazovce stiskne tlačítko pro vypnutí zařízení. Na displeji je zobrazen dialog pro potvrzení operace
- `STATE_RESTART` (**Restart systému**) - v tomto stavu se aplikace nachází, pokud uživatel na výchozí obrazovce stiskne tlačítko pro restart zařízení. Na displeji je zobrazen dialog pro potvrzení operace
- `STATE_INIT_ERROR` (**Chyba inicializace**) - stav, který nastává v situaci, když se během inicializace aplikaci nepodaří v časovém limitu získat

seznam dostupných skriptů. V tomto stavu je na displeji zobrazen dialog, který umožňuje operaci zopakovat

- `STATE_SCRIPT_ERROR` (**Chyba spuštění skriptu**) - stav, který nastává v situaci, když aplikace během spouštění skriptu neobdrží odpověď v časovém limitu. V tomto stavu je na displeji zobrazen dialog, který uživateli umožní operaci opakovat, či se navrátit do výchozího stavu (`STATE_MENU`)

Pro každý stav/obrazovku existuje v aplikaci příslušná třída, která má na starosti správné vykreslení své dané obrazovky. Aplikace dále obsahuje hlavní třídu `MainWindow`, která zde obstarává veškerou logiku systému a přepíná mezi různými stavy/obrazovkami aplikace. Třída `MainWindow` dědí ze třídy `QMainWindow` a její obsah tvoří widget `QStackedWidget`, který obsahuje widgety ostatních tříd, mezi kterými aplikace přepíná.

4.3.3 Inicializace aplikace

Inicializace aplikace probíhá následujícím způsobem:

1. Aplikace provede registraci na odběr událostí `scriptListUpdate`
2. Aplikace přechází do stavu `STATE_INIT`
3. Aplikace se pokusí prostřednictvím REQ získat aktuální seznam skriptů
4. Aplikace zobrazí výchozí nabídku s volbou skriptu. Přechází do stavu `STATE_MENU`

Poté, co je aplikace zinicializována, je připravena jednak reagovat na uživatelské akce nebo na případné přijetí nového seznamu skriptů (událost `scriptListUpdate`).

4.3.4 Vývoj GUI

Návrh jednotlivých obrazovek byl vytvořen pomocí nástroje QtCreator, což je nástroj, který slouží k tvorbě Qt projektů a mimo jiné umožňuje jednoduše vytvořit návrh uživatelského rozhraní. Díky tomu je možné z velké části oddělit tvorbu UI od aplikační logiky.

4.3.4.1 Návrh obrazovek

QtCreator ukládá návrhy komponent (widgetů) jako textový UI soubor (s příponou .ui) ve formátu XML. Takový soubor je během sestavování projektu pro potřeby kompilace převeden na klasický .cpp a .h soubor. Tímto způsobem byly vytvořeny návrhy všech obrazovek. Qt umožňuje ke každému .ui souboru vytvořit obalující třídu, tedy nový hlavičkový a zdrojový soubor, který má

přímý přístup k prvkům vytvořeným v grafickém návrháři. Tímto způsobem šlo velmi pohodlně oddělit prezentační část (reprezentovanou .ui souborem) od aplikační logiky (reprezentovanou C++ třídou) u jednotlivých obrazovek.

4.3.4.2 Zpracování výstupu skriptů

Pro zobrazení výstupu skriptů byl použit widget `QTextBrowser`, který díky nastavení `acceptRichText` umožňuje, kromě obyčejného textu zobrazit i text obohacený o HTML tagy. Bohužel, widget podporuje pouze určitou podmnožinu HTML4 a CSS (`http://doc.qt.io/qt-5.8/richtext-html-subset.html`). Tím jsou tedy možnosti uživatelských skriptů částečně omezeny. Nicméně i s dostupnými možnostmi je možné vytvořit přehlednou prezentaci, to lze vidět na obrázku 4.3, kde je zobrazen výstup z ukázkového skriptu pro předpověď počasí.

Vzhledem k omezeným možnostem widgetu `QTextBrowser` při zpracování HTML jsem pátral i po jiných možnostech jak pomocí Qt zobrazit HTML dokument. Od verze 5.4 existuje v Qt modul s názvem Qt WebEngine, který poskytuje sadu tříd určených k vykreslování HTML, XHTML a SVG dokumentů. Tyto třídy podporují také CSS kaskádové styly a JavaScript. Použití modulu Qt WebEngine vlastně zpřístupňuje plnohodnotný webový prohlížeč, jelikož jádro modulu je postaveno přímo na projektu Chromium. [26] Použitím tohoto modulu by tedy bylo možné vykreslit libovolný HTML kód, včetně HTML5. Modul Qt WebEngine ovšem není standardně součástí knihovny. Tedy při kompilaci ze zdrojových kódů, jako v tomto případě, nebude modul sestaven. Zdrojové soubory modulu Qt WebEngine se nacházejí v adresáři qtwebengine v původním repozitáři Qt knihovny. Pokusil jsem se tedy modul sestavit obdobným způsobem jako původní knihovnu, nicméně zde se při kompilaci pomocí cross kompileru vyskytly komplikace a modul se mi tedy pro zařízení Raspberry Pi sestavit nepodařilo.

4.3.5 Práce s aplikací

Obsluha aplikace je velice jednoduchá. Na výchozí obrazovce, kterou je možno vidět na obrázku 4.2, je vidět aktuálně zvolený skript. Pomocí bočních tlačítek je možno v tomto seznamu listovat. Uprostřed obrazovky je zobrazen titulek a popis aktuálního skriptu. Obě tyto hodnoty se nastavují u jednotlivých skriptů v konfiguračním rozhraní zařízení. Tlačítka na spodní straně displeje slouží k restartu či vypnutí samotného zařízení. Po jejich stisknutí zobrazí aplikace nejprve dialog, kde je nutné zvolenou akci potvrdit. Po kliknutí kamkoliv do oblasti titulku a popisu skriptu se aplikace pokusí tento skript spustit.

Pokud vše proběhne v pořádku, zobrazí se na obrazovce výstup skriptu, což je znázorněno na obrázku 4.3. Výsledná prezentace může být samozřejmě větší, než je velikost displeje. Z toho důvodu umožňuje aplikace na této obrazovce jednoduše skrolovat po obou osách (Pokud je v dané ose pro skrolování

prostor). Na obrázku je také vidět spodní lišta s nabídkou. Tato lišta se zobrazí poté, co uživatel klikne kamkoliv na displej (kromě odkazu). Další stisknutí či skrolování lištu opět schová, jelikož by mohla překážet samotné prezentaci. Lišta nabízí tlačítka pro uzavření skriptu (přechod aplikace na obrazovku s nabídkou skriptů), okamžitou aktualizaci (znovunačtení), a tlačítka pro změnu aktuálně zobrazeného skriptu (listování v seznamu skriptů).

Obrázek 4.2: Hlavní menu aplikace

Obrázek 4.3: Ukázka spuštěného skriptu

4.4 Správce skriptů

Správce skriptů je samostatná komponenta, která vznikla za účelem spouštění uživatelských skriptů a předávání jejich výstupu ostatním službám. Služba pracuje primárně nad ZeroMQ sockety, tedy sockety otevřenými a udržovanými pomocí knihovny ZeroMQ. Zde služba čeká jednak na příchozí požadavek o spuštění skriptu a dále také na případný příchod události `scriptListUpdate`, která nese aktualizovaný seznam uživatelských skriptů.

4.4.1 Inicializace služby

Na odběr událostí `scriptListUpdate` se Správce skriptů registruje během inicializace u komponenty (služby) Správce webového rozhraní. Struktura této události je popsána v sekci 4.2.2.3. Při jejím přijetí služba odstraní veškeré naimportované skripty a poté naimportuje nové dle nového seznamu. Registrace probíhá prostřednictvím modulu `zmq`, což je port knihovny ZeroMQ pro jazyk Python. Inicializace služby probíhá následujícím způsobem.

1. Služba si prostřednictvím služby Správce webového rozhraní vyžádá aktuální seznam skriptů.
2. Služba vytvoří serverový REP socket pro vyřizování požadavku na spouštění webových skritpů.
3. Služba vytvoří serverový REP socket pro vyřizování požadavku na spuštění skriptů pro připojený displej. Důvodem pro vytvoření dvou různých socketů je, že struktura přenášených zpráv se v obou případech liší. Stejně tak se liší následné zpracování požadavku.
4. Služba se registruje na odběr událostí `scriptListUpdate` u služby Správce webového rozhraní. Za tímto účelem vytváří socket typu SUB, který se na druhé straně připojuje na socket typu PUB.
5. Služba přechází do výchozího stavu. Zde pomocí funkce `poll` z modulu `zmq` sleduje všechny otevřené sockety a v případě příchozí události, či požadavku spouští obslužné funkce. Po jejich vyřízení opět přechází do výchozího stavu.

Komunikace a obsluha požadavků na spuštění skriptu je primární činností této komponenty a je blíže popsána v následujících sekcích. Veškerá tato komunikace probíhá opět formou výměny zpráv, které jsou ve formátu JSON.

4.4.2 Skripty pro displej

V této sekci se věnuji popisu komunikace mezi službou Správce skriptů a Správcem displeje. Tato komunikace vždy začíná na straně Správce displeje

vytvořením požadavku na spuštění skriptu. Správce skriptů následně spustí funkci `start` s parametrem `link`, který byl předán společně s požadavkem. Návratová hodnota funkce `start` je předána jako odpověď. Struktura požadavku a odpovědi je následující:

- Požadavek

```
{
    scriptName: <String>
    link: <String>
}
```

- Odpověď

```
{
    payload: <String>
}
```

Atribut požadavku `scriptName` nese jméno skriptu, který má být vykonán. Pokud byl požadavek na skript vyvolán stiskem odkazu na displeji (tag `<a>`), nese atribut link hodnotu atributu `href` tagu `<a>`. Pokud byl požadavek vykonán z důvodu pravidelné aktualizace prezentace, či přímo uživatelem, bude hodnota atributu `link` obsahovat stejnou hodnotu, se kterou byl spuštěn naposledy (z důvodů popsaných v sekci 3.9.2). Jinak obsahuje atribut `link` prázdný řetězec.

Atribut odpovědi `payload` obsahuje řetězec odpovídající výstupu požadovaného skriptu. Respektive řetězec odpovídající návratové hodnotě funkce `start`.

4.4.3 Webové skripty

V této sekci se věnuji popisu komunikace mezi službou Správce skriptů a Správcem webového rozhraní, respektive jeho částí odpovědnou za vyřizování webových uživatelských skriptů. Tato komunikace opět vzniká na straně Správce webového rozhraní vytvořením požadavku na spuštění skriptu. Zde se struktura zpráv liší od předchozího případu.

- Požadavek

```
{
    scriptName: <String>
    params: <Object>
}
```

- Odpověď

```
{
    statusCode: <Number>,
    contentType: <String>,
    payload: <String>
}
```

Atribut params nese seznam parametrů předaných pomocí HTTP metody GET při přístupu na webové rozhraní skriptu. Hodnotu tohoto atributu tvoří JSON objekt, kde je klíčem název parametru a jeho hodnotou je hodnota tohoto parametru. Tedy například při vyvolání požadavku `/webScript/weather?date=1494004514` na webové rozhraní zařízení, tedy požadavku uživatele na provedení skriptu weather.py, bude JSON podoba zprávy následující.

```
{
    "scriptName": "weather"
    "params": {
            "date": 1494004514
    }
}
```

Obsah odpovědi na požadavek přesně odpovídá návratové hodnotě funkce `start`. Tato je popsána v sekci (3.9.1).

4.5 Konfigurace zařízení

4.5.1 Inicializace systému

Cílem bylo vytvořit zařízení, které bude sloužit k jednomu konkrétnímu účelu. Očekávané chování je tedy takové, že po jeho spuštění se spustí i veškeré nutné aplikace a zařízení se tedy automaticky dostane do stavu, kdy je na displeji zobrazena volba skriptů a běží webové rozhraní. Proto bylo třeba nastavit systém tak, aby došlo ke spuštění všech komponent po startu automaticky bez nutnosti zásahu uživatelem. To znamenalo spustit všechny tři komponenty jako služby.

Služby se v linuxovém operačním systému liší od běžných aplikací. Pokud od naší aplikace očekáváme, že se bude chovat jako služba, znamená to, že poběží nezávisle na relaci, odkud byla uživatelem spuštěna. Například pokud uživatel spustí službu z terminálu, služba musí běžet i poté, co se uživatel odhlásí. Spustí-li uživatel běžnou aplikaci, tak vznikne nový proces a jeho rodičovským procesem je právě proces, který ho spustil. Nový proces má také přiřazený kontrolní terminál, který je stejný jako u rodičovského procesu. Na tento terminál jsou typicky nasměrovány souborové deskritpory 0,1 a 2 procesu, tedy standardní vstup, výstup a chybový výstup. Aplikaci je možné spustit na pozadí přidáním znaku „&“ za spouštěný příkaz. Tím ale nedojde

k požadovanému chování. Takto spuštěná aplikace je stále ve stejné relaci jako předtím, to znamená, že po odhlášení uživatele dojde k ukončení jeho relace a systém zároveň ukončí všechny procesy z této relace. Jednou z možností, jak dosáhnout toho, že aplikace poběží i po ukončení uživatelské relace, je spustit ji s pomocí nohup a na pozadí, tedy příkazem `nohup <aplikace> &`. Toto ale není ideální řešení. Pro typickou službu v linuxovém operačním systému platí, že jejím rodičovským procesem je proces init, tedy proces s PID 1. Proces běžící služby také není svázán s žádným kontrolním terminálem a běží ve vlastní relaci. Dosažení takového stavu svépomocí znamená provést řadu operací, které, mimo jiné, zahrnují dvojité volání funkce fork a volání funkce setsid. Tím je ve finále dosaženo toho, že proces běží nezávisle na původní relaci a tedy její ukončení na něj nemá vliv. Tento postup je nicméně poměrně komplikovaný a nepohodlný. Naštěstí existuje řešení, které tento proces poměrně dost usnadňuje. Tímto mám na mysli službu systemd. [27]

Systemd je sada základních stavebních bloků systému Linux. Jedná se o správce služeb, který běží s PID 1 (tedy proces init) a obstarává spouštění zbytku systému. Systemd init, na rozdíl od klasického init systému, spouští procesy přímo a tedy není nutné, aby služba prováděla výše popsané operace jako volání funkce fork, či setsid. Tím je díky Systemd možné, aby se psaní démonů nelišilo od psaní klasických aplikací. [27] Toho bylo využito i zde. Aby mohl proces init systému Systemd spustit aplikaci jako službu (démona), je nutné pro ni vytvořit konfigurační soubor. Bylo tedy nutné vytvořit konfigurační soubor pro všechny tři vzniklé služby. Následující konfigurační soubor je konfigurace pro spuštění Správce displeje:

```
[Unit]
Description=Ovladac dotikoveho displeje
After=web-service.service

[Service]
ExecStart=/opt/bin/display -platform linuxfb
Restart=always
RestartSec=5

[Install]
WantedBy=multi-user.target
```

Direktiva After v sekci Unit udává, že má být tato služba spuštěna až poté, co bude spuštěna služba web-service (Správce webového rozhraní). Toto je pro správné spuštení nezbytné, jelikož ihned po spuštění se, prostřednictvím knihovny ZeroMQ, tato služba snaží kontaktovat právě službu web-service, aby získala aktuální seznam uživatelských skriptů. Takto je zajištěno to, že po spuštění již služba web-service poběží a tedy bude připravená seznam skriptů poskytnout.

Direktiva ExecStart v sekci Service udává, jaký program a s jakými pa-

rametry má být spuštěn. Pokud by spuštění vyžadovalo nějaký komplexnější postup, tak by zde byl typicky příkaz na spuštění příslušného init skriptu. V tomto případě to ale není nutné.

Direktiva Restart udává, že v případě ukončení služby (např. z důvodu pádu aplikace), se má služba opět spustit. Následující direktiva RestartSec udává, za jak dlouho od ukončení k tomu dojde.

Nakonec direktiva WantedBy v sekci Install říká, že tato služba musí být spuštěna pro dosažení stavu multi-user.target. Multi-user.target je v podstatě obdoba runlevelu 3 z klasického init systému. Posledním řádkem tedy říkáme, kdy se má služba spustit (a zda vůbec) během inicializace systému. Tento řádek není povinný, pokud bychom ho neuvedli, k automatickému spuštění služby by nedošlo. Ovšem stále by bylo možné službu spustit příkazem systemctl.

Konfigurační soubor služby pro správu skriptů je velice podobný. I tento má v direktivě After uvedenou závislost na službu web-service, jelikož i zde dochází ihned po startu k inicializaci seznamu skriptů. Tím je tedy dané pořadí spuštěných služeb, jako první se spustí služba web-service a zbylé služby až po ní (na jejich vzájemném pořadí nezáleží). Takto by se mohlo tedy zdát, že je zajištěno to, že při startu služeb pro správu displeje a pro správu skriptů je již služba web-service připravena a tedy, že bude reagovat na požadavky o seznam skriptů. Ovšem není tomu tak. Poté co Systemd spustí službu web-service, již se dál nestará, zda je služba řádně zinicializována (sám od sebe ani nemůže) a ihned spouští její zavislosti. Nicméně po spuštění služby web-service ještě jistou dobu trvá než dojde ke spuštění webového serveru (HTTP i HTTPS) a až poté služba vytváří sockety pro příjem požadavků prostřednictvím ZeroMQ knihovny. Sysemd ale dokáže řešit i tento problém. Využívá k tomu volání `sd_notify`, kterým může spouštěný proces systému sdělit, že je již připravena a je tedy možné spustit další závislé služby. Služba web-service nevolá přímo `sd_notify`, ale v okamžiku, kdy je plně zinicializována, zavolá příkaz `systemd-notify` s parametrem `--ready`. `systemd-notify` je pouze wrapper pro volání funkce `sd_notify`. Aby Systemd věděl, že má čekat na volání funkce `sd_notify`, je nutné do konfigurace služby vložit nastavení `Type=notify`. Kompletní konfigurace služby web-service je následující:

```
[Unit]
Description=Node.js webove rozhrani

[Service]
Type=notify
ExecStart=/usr/bin/node /opt/web_service/index.js pi
Restart=always
NotifyAccess=all
TimeoutStartSec=120
```

```
[Install]
WantedBy=multi-user.target
```

Nastavení `NotifyAccess=all` udává, že jsou akceptována všechna volání `sd_notify` z dané kontrolní skupiny. Zde je nutné nechat hodnotu All, jelikož `sd_notify` v tomto případě nevolá přímo samotná služba, ale, jak je popsáno výše, využívá k tomu příkaz `systemd-notify --ready`, což je vlastně úplně nový proces.

4.5.2 Připojení k síti

Aby bylo zařízení prakticky použitelné, je nutné, aby bylo schopné se, pokud možno samo, připojit k síti. Zařízení obsahuje jednak konektor RJ-45 pro připojení k ethernetu a dále je k němu připojen wifi usb dongle pro připojení k bezdrátové síti. Jak již bylo řečeno výše, připojení k síti wifi se nastavuje přímo prostřednictvím webového konfiguračního rozhraní. Nicméně aby bylo možné k rozhraní přistoupit, je nutné, aby zařízení již k nějaké síti připojené bylo. V takovém případě tedy zbývá připojit zařízení přes rozhraní ethernet. K tomuto účelu je zařízení nastaveno tak, aby se k síti prostřednictvím rozhraní `eth0` připojovalo automaticky s konfigurací získanou prostřednictvím DHCP klienta. Díky tomu je tedy možné zařízení bez problému připojit k běžnému domácímu routeru.

Počítal jsem ale i s variantou, že z nějakého důvodu nebude možné/pohodlné zařízení k routeru pomocí kabelu připojit. Tedy přesněji, že zařízení nebude možné připojit do sítě, kde běží nějaký DHCP server. Z toho důvodu jsem vytvořil nové virtuální rozhraní `eth0:1`, které má napevno nastavenou adresu na hodnotu 192.168.0.1/24. Takto je tedy možné zařízení připojit přímo k jinému počítači a do URL prohlížeče zadat právě adresu 192.168.0.1. Tím se dostaneme na webové konfigurační rozhraní, kde je možné nastavit připojení k wifi síti.

Výše popsaná konfigurace je provedena přidáním následujících řádků do souboru /etc/network/interfaces.

```
iface eth0 inet dhcp

auto eth0:1
iface eth0:1 inet static
address 192.168.0.1
netmask 255.255.255.0
```

Kapitola 5

Ukázkové skripty

V této kapitole se věnuji popisu několika vybraných ukázkových skriptů, na kterých byl celý systém testován a které rovněž demonstrují různé možnosti zařízení.

5.1 Aktuální předpověď počasí

Tento skript slouží k zobrazení aktuálního počasí v Praze. Skript využívá rozhraní `api.openweathermap.org`, což je veřejně dostupné API, které nabízí strukturovanou formou přehled jak aktuálního počasí, tak i předpověď na následující dny. Rozhraní funguje nad protokolem HTTP, tedy stačí provést standardní HTTP požadavek metodou get. Data jsou předávána v JSON formátu, což umožňuje jejich pohodlné zpracování. Výstup skriptu je zobrazen na obrázku 5.1.

Obrázek 5.1: Aktuální předpověď počasí pro Prahu

Smyslem tohoto skriptu je demonstrovat možnosti uživatelských skriptů. Jak je vidět, do výsledné prezentace byla vložena ikona odpovídající aktuální předpovědi. K tomuto účelu je použit standardně tag `<img>`, nicméně není možné použít přímo adresu vzdáleného zdroje (obrázku). Chceme li do výsledné prezentace vložit obrázek, je nutné aby se nacházel lokálně na zařízení, tedy zde byl obrázek nejprve stažen do adresáře /tmp. Obrázek se poté vloží následujícím způsobem: `<img src="file://tmp/icon.png">`

5.2 Předpověď počasí na následující dny

Tento skript slouží pro předpověď počasí na následující dny. Skript je podobný předchozímu, nicméně na úvodní obrazovce nejprve nabídne volbu dne, pro který nás předpověď zajímá. Skript využívá stejný zdroj jako předchozí (`api.openweathermap.org`). Výstup skriptu je zobrazen na obrázku 4.3.

Obrázek 5.2: Předpověď počasí pro Prahu

Skript demonstruje možnosti interakce s uživatelem. Skript vytváří dvě obrazovky, první nabízí volbu dne a druhá již samotnou předpověď. Interakce probíhá pomocí HTML odkazu (tag <a>). Způsob, jakým tato interakce funguje, je blíže popsán v sekci 3.9.2.

5.3 Odjezdy MHD

Tento skript zobrazuje nejbližší časy odjezdů zvolených linek MHD k aktuálnímu času a jasně definovanému místu (napevno ve skriptu). Skript pracuje s daty získanými z webové stránky `jizdnirady.idnes.cz`. Skript opět nabízí nejprve stránku s volbou autobusové linky a až následně přehled odjezdů.

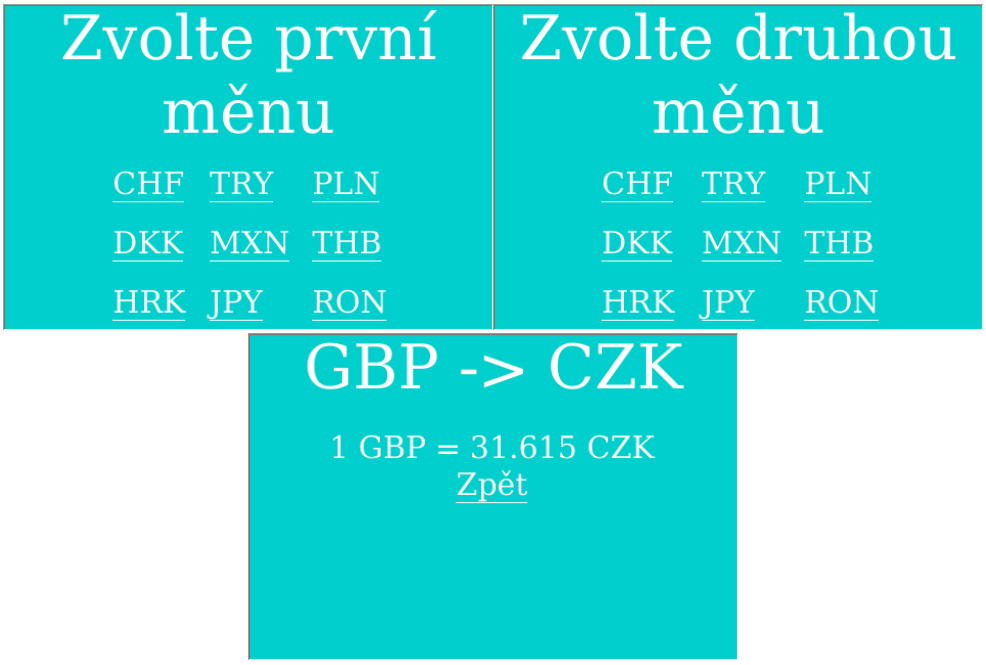
Tento skript demonstruje, jakým způsobem je možné zpracovat webovou HTML stránku za účelem získání vstupních dat přímo z HTML dokumentu. V tomto se ukázka liší od předchozích. Předchozí skripty pracovaly vždy s nějakým API, tedy rozhraním, které svá data zpřístupňuje strukturovanou formou, jelikož slouží právě pro dodatečné zpracování. Naproti tomu účelem

Obrázek 5.3: Odjezdy MHD

HTML dokumentu je přímo prezentace dat, tedy není to zcela pohodlný formát pro další zpracování. Skript pro zpracování vstupního HTML dokumentu využívá především regulární výrazy.

5.4 Kurzy měn

Tento skript slouží k zobrazení aktuálního vzájemného kurzu dvou zvolených měn. Jedná se o další interaktivní skript, kde si uživatel na prvních dvou obrazovkách volí měnu. Následně je na displeji zobrazen aktuální kurz těchto dvou měn. Skript čerpá data z veřejně dostupného api `http://fixer.io/`.



Obrázek 5.4: Přehled kurzů měn

5.5 Adresář

Tento skript demonstruje práci s lokálně uloženými daty. Skript nepřistupuje k internetu, ale pracuje s lokálně uloženým souborem, který obsahuje jména lidí a příslušné údaje. Zdrojový soubor, ale i ikonka mailu a telefonu byly do zařízení vloženy prostřednictvím webového konfiguračního rozhraní (sekce Soubory).

Obrázek 5.5: Výstup skriptu Adresář

Závěr

V rámci této diplomové práce vznikl nástroj pro zpracování a prezentaci dat z internetu na základě uživatelských skriptů, který je určený pro platformu Raspberry Pi. K prezentaci dat slouží připojený dotykový displej a webové rozhraní.

Pro psaní samotných uživatelských skriptů byl zvolen skriptovací jazyk Python. Skripty jsou rozděleny na dva druhy a to skripty pro displej a skripty pro webové rozhraní. Skripty pro displej mohou data prezentovat jako jednoduchou HTML stránku (systém podporuje podmnožinu HTML4 a CSS stylů), která bude vykreslena na připojeném displeji. Tyto skripty také podporují omezenou formu interakce s uživatelem, pomocí vkládání odkazů do výsledné prezentace.

Systém se skládá ze tří základních komponent - Správce webového rozhraní, Správce displeje a Správce skriptů. Správce skriptů je komponenta, která vznikla pouze za účelem vykonávání uživatelských skriptů. Díky tomu, že je oddělena od zbytku systému, je možné v budoucnu tuto komponentu snadno nahradit jinou, bude-li to třeba. Vzájemné propojení všech součástí systému zajišťuje distribuovaný nástroj ZeroMQ.

Zařízení nabízí jednoduché webové konfigurační rozhraní, pomocí kterého je možné spravovat seznam skriptů.

Výsledek práce ukázal možnosti využití počítače Raspberry Pi jako nástroje pro prezentaci dat. Práce byla vyvíjena (dle zadání) s důrazem na data získaná z internetu, ale do budoucna by mohlo být zařízení uzpůsobeno k získávání dat i z jiných zdrojů.

Literatura

[1] Carter, N.: rpi-weather. [online], 2016, [cit. 2017-05-01]. Dostupné z: https://github.com/caternuson/rpi-weather

[2] Piney: Raspberry Pi Wall Mounted Google Calendar.

[3] BeagleBone Black. [online], březen 2017, [cit. 2017-05-01]. Dostupné z: https://beagleboard.org/black

[4] Beagleboard:Android. *Embedded Linux Wiki [online]*, listopad 2014, [cit. 2017-05-01]. Dostupné z: http://elinux.org/Beagleboard:Android

[5] ODROID-C2. [online], [cit. 2017-05-01]. Dostupné z: http://www.hardkernel.com/main/products/prdt_info.php?g_code=G145457216438

[6] Raspberry Pi 3 Model B. [online], [cit. 2017-05-01]. Dostupné z: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[7] UDOO X86: The Most Powerful Maker Board Ever. *Kickstarter [online]*, listopad 2016, [cit. 2017-05-01]. Dostupné z: https://www.kickstarter.com/projects/udoo/udoo-x86-the-most-powerful-maker-board-ever

[8] UDOO X86. [online], [cit. 2017-05-01]. Dostupné z: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[9] Banana Pi vs. Raspberry Pi. *Banana Pi [online]*, [cit. 2017-05-01]. Dostupné z: http://www.bananapi.org/2014/05/is-banana-pi-clone-of-raspberry-pi.html

[10] SBC. *Banana Pi [online]*, [cit. 2017-05-01]. Dostupné z: http://www.bananapi.org/2014/05/is-banana-pi-clone-of-raspberry-pi.html

[11] Bush, S.: Dongle computer lets kids discover programming on a TV. *Electronics Weekly [online]*, březen 2011, [cit. 2017-05-01]. Dostupné z: https://www.electronicsweekly.com/market-sectors/embedded-systems/dongle-computer-lets-kids-discover-programming-on-a-2011-05/

[12] *BOOTPARAM(7) Linux Programmer's Manual*. květen 2017. Dostupné z: http://man7.org/linux/man-pages/man7/bootparam.7.html

[13] Soyding, C.: RASPBERRY PI BOOT SEQUENCE. [online], květen 2013, [cit. 2017-05-01]. Dostupné z: https://en.wikipedia.org/wiki/File:Atomic_force_microscope_block_diagram.svg

[14] Welcome to Raspbian. [online], [cit. 2017-05-01]. Dostupné z: http://www.http://raspbian.org/

[15] Raspbian FAQ. [online], [cit. 2017-05-01]. Dostupné z: http://www.http://raspbian.org/RaspbianFAQ

[16] Debian Wiki team: RaspberryPi. [online], listopad 2016, [cit. 2017-05-01]. Dostupné z: http://www.http://raspbian.org/RaspbianFAQ

[17] Rouse, M.: Node.js. *WhatIs [online]*, duben 2012, [cit. 2017-05-01]. Dostupné z: http://whatis.techtarget.com/definition/Nodejs

[18] The Node.js Event Loop, Timers, and process.nextTick(). *Node.js [online]*, [cit. 2017-05-01]. Dostupné z: https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/

[19] Tranter, J.: Using Qt with Alternative Programming Languages - Part 1. *ICS [online]*, srpen 2015, [cit. 2017-05-01]. Dostupné z: https://www.ics.com/blog/using-qt-alternative-programming-languages-part-1

[20] Using a Designer UI File in Your Application. *Qt Documentation [online]*, 2016, [cit. 2017-05-01]. Dostupné z: http://doc.qt.io/qt-4.8/designer-using-a-ui-file.html

[21] Hohpe, G.: Hub and Spoke [or] Zen and the Art of Message Broker Maintenance. *Enterprise Integration Patterns [online]*, listopad 2003, [cit. 2017-05-01]. Dostupné z: http://www.enterpriseintegrationpatterns.com/ramblings/03_hubandspoke.html

[22] Hintjens, P.: ØMQ - The Guide. [online], [cit. 2017-05-01]. Dostupné z: http://zguide.zeromq.org/page:all

[23] Chatriot, L.: NeDB. [online], 2016, [cit. 2017-05-01]. Dostupné z: `https://github.com/louischatriot/nedb`

[24] *wpa_supplicant(8) - Linux man page*. Dostupné z: `https://linux.die.net/man/8/wpa_supplicant`

[25] Qt for Embedded Linux. *Qt Documentation [online]*, 2017, [cit. 2017-05-01]. Dostupné z: `http://doc.qt.io/qt-5/embedded-linux.html`

[26] Qt WebEngine Overview. *Qt Documentation [online]*, 2017, [cit. 2017-05-01]. Dostupné z: `http://doc.qt.io/qt-5/qtwebengine-overview.html`

[27] Vyskočil, M.: Systemd – psaní unixových démonů. *ABC Linuxu [online]*, červen 2011, [cit. 2017-05-01]. Dostupné z: `http://www.abclinuxu.cz/clanky/systemd-psani-unixovych-demonu`

Příloha **A**

Seznam použitých zkratek

ABI Application binary interface

XML Extensible markup language

GUI Graphical user interface

GCC GNU compiler collection

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

API Application Programming Interface

REST Representational State Transfer

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `project`
  - `display` ..... zdrojové kódy správce displeje
  - `web_service` ..... zdrojové kódy správce webového rozhraní
  - `script_service` ..... zdrojové kódy správce skriptů
  - `instalace` ..... soubory a návody pro instalaci aplikace
  - `skripty` ..... ukázkové skripty
- `thesis`
  - `latex` ..... adresář se zdrojovou formou práce ve formátu LaTeX
  - `DP_Kuzel_jakub_2017.pdf` ..... text práce ve formátu PDF
  - `zadani.pdf` ..... zadání této práce