SEMINÁRNÍ PRÁCE

Předmět: Programování a výpočetní technika

Monitoring terária

Terrarium monitoring

Škola: Gymnázium Teplice, Čs. dobrovolců 11

Kraj: Ústecký
Vypracoval: Prokop Parůžek, 7.A
Konzultant: Ing. Věra Minaříková

Teplice 2021

Prohlášení

Prohlašuji, že jsem svou seminární práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v seznamu vloženém v práci.

Prohlašuji, že tištěná verze a elektronická verze práce jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) v platném znění.

V Teplicích dne ....

Prokop Parůžek

Poděkování

Děkuji ... za ... (upravte makro `\podekovani{}`).

Prokop Parůžek

Anotace: Cílem práce je vytvořit automatický systém na sledování teploty a vlhkosti a dalších údajů v teráriu s masožravými rostlinami. Zpřístupnit naměřené údaje online, v podobě grafů, aby uživatel mohl v reálném čase sledovat jak se jeho kytičkám daří. Zároveň je kladen důraz na snadnou rozšiřitelnost o další naměřené hodnoty, či o úplně nové senzory, místnosti.

Klíčová slova: měření, IoT,

Annotation:

Key words: measure, IoT,

Obsah

Úvod

Už potřetí zahajuji svůj pokus pěstovat masožravé rostliny, který zatím vždy skončil jejich úhynem. Z toho důvodu jsem se rozhodl začít sledovat prostředí v teráriu, kde je pěstuji, abych mohl v případě úhynu určit z jakého důvodu uhynuly. Přehřáli se, umrzli, uschly... Většinou z důvodu mé nepřítomnosti, kdy jsem je nemohl kontrolovat. Avšak mnohem radši bych byl, kdyby se mi pomocí naměřených údajů podařilo udržet prostředí ve kterém prosperují a v případě náhlé změny mohl zasáhnout v krajním případě i vzdáleně.

Cílem mé práce je navržení systému pro měření v podstatě libovolných hodnot, jejich agregování na jednom místě, s možností zobrazení aktuálních dat, či jejich průběhu v minulosti, či navázáním různých alarmů na kritické hodnoty. Hodnoty by uživatel kontroloval s využitím webové aplikace, které zároveň zajistí snadnou použitelnost na mnoha platformách a přístupnost takřka na celém světě, tedy tam kde je internet.

Výsledkem práce bude samotná realizace řešení, od výběru hardwaru a dalších věcí jako je databáze... po samotné sestavení měřícího zařízení, jeho naprogramování a naprogramování aplikace na zobrazení naměřených dat. Výsledný produkt by měl být snadno použitelný a rozšiřitelný o další funkce, možný budoucí vývoj je až aplikace na řízení tzv. chytrého domu. Z tohoto důvodu bude kladen důraz i na zabezpečení, pro zamezení neoprávněného přístupu. Z důvodů urychlení a zlevnění vývoje, nebudu vždy používat nejvhodnější, ale nejdostupnější řešení tj. to které už znám, či u hardwaru to co mám doma.

Kapitola 1

Požadavky na řešení

První požadavek se bude týkat měření. Měřit chci teplotu a vlhkost v teráriu, když bude zvolený hardware umět i něco jiného, klidně to použiji, ale hlavní požadavek je na tyto dvě veličiny. Ohledně frekvence měření chci zachytit denní trendy, ale nepotřebuji data z každé minuty.

Další z požadavků je na ukládání dat. Když už je změřím, tak je chci mít vždy uložená, tedy i při výpadku internetu a podobně. Při výpadku proudu nic nezměřím, takže to není třeba řešit. Co se týče vzdáleného ukládání, nepovažuji za důležité, aby se všechna data propsala do cloudu, tedy i v případě nějakého výpadku se odeslala data co jsem změřil, ale neodeslal. Když přijdu o jedno měření během krátkého výpadku, je mi to jedno.

Data mám uložena, co s nimi budu dělat? No tak v podstatě bych nemusel dělat nic, pouze si je zobrazit, což je to co požaduji. Avšak hezká by byla možnost zobrazit si nějaké pokročilejší statistiky. Takže volitelně přidávám požadavek na zobrazení různých časových rámců a statistiky k onomu časovému období, například průměrná, nejvyšší, nejnižší či medián teploty a vlhkosti. Případně různé tendence, derivace, co bude v možnostech vybraného nástroje.

Nároky na uživatelské rozhraní v podstatě nemám. Pro správu senzorů je nepotřebné. Stejně moc obměňovat, či přidávat nebudu, a i kdyby Stejně si budu muset napsat obslužný program pro ten či onen. Udělat nějaký obalující systém pro více senzorů, či knihovnu není mým cílem. Pro zobrazení hodnot je důležité, ale ohledně nároků mi bude stačit velmi jednoduché, pokud možno jediná stránka. Avšak případnému rozšíření se nebráním. Další části u nichž by bylo třeba komunikovat s uživatelem mě nenapadají. Možná správa uživatelů s přístupem, ale vzhledem k tomu, že to dělám pro sebe bych to vynechal.

K bezpečnosti bych rád zmínil toto. Bylo by dobré mít komunikaci po celé komunikační trase šifrovanou, ale dokud řešení neobsahuje ovládání čehosi, není to úplně nezbytné. Případný útočník by si tak maximálně zjistil vlhkost... v teráriu nebo by se mohl teoreticky vydávat za teploměr a kazit mi data. To by mohl být problém, takže pokud nevyžaduji šifrování, ověření toho kdo posílá data požaduji určitě. Zabezpečení však považuji za důležité u samotné webové stránky, která bude vystavena veřejně. Ne že by mi vadilo, že někdo sleduje jak se mají mé kytičky, ale

mohlo by se z toho dát odvodit, že je nezalévám tj. jsem pryč a v bytě nikdo není.

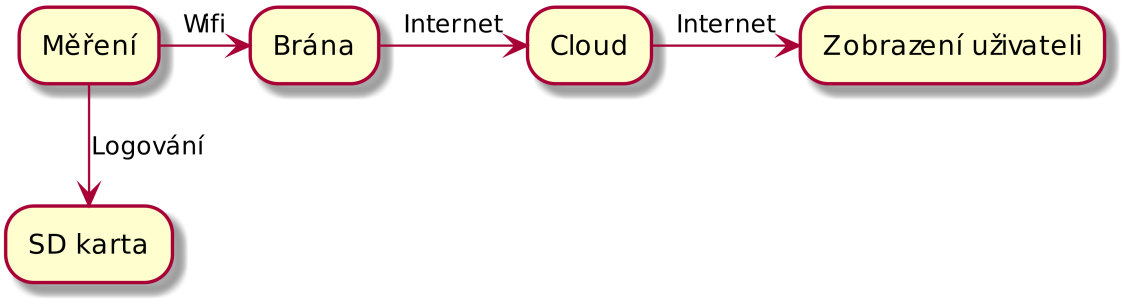
Kapitola 2

Analýza problému

Ohledně měření bych zatím zmínil jen frekvence o které myslím myslím, že pro mé účely bude stačit měřit jednou za čtvrt hodiny, to je 96 měření za den. Neodchytím tím sice drobné výkyvy v rámci minut, ale cílem je zachytit dlouhodobý průběh hodnot v rámci dne, kdy mne tak drobné výkyvy nezajímají. Pro tento účel by se čtvrt hodiny mohlo zdát možná až jako příliš často, ale já bych to tak nechal z důvodu zjemnění denních grafů, přeci jenom graf se třeba 12 hodnotami nevypadá úplně nejlépe, a také z důvodu odchycení případných chyb měření nebo náhlých změn, například při zalévání atd.

Co se týče uložení dat, abych zajistil stoprocentní jistotu, že se data uloží, budu je ukládat přímo v místě měření, tím myslím, že počítač, který bude mít na starosti měřit, bude zároveň naměřená data hned ukládat na SD kartu, nebo tak někam. Bude to takový log měření, který mi umožní obnovit data nezapsaná do cloudu z důvodu nějaké chyby, případně mi umožní zjišťovat kde vlastně chyba nastala, a může pomoci i s řešením.

Teď se dostávám k samotnému data flow, tedy jak a kam mi potečou data co naměřím. Začnu u senzoru, ten změří data a pošle je do obslužného počítače, to může být v podstatě cokoli se schopností ovládat senzor,ukládat data a schopností poslat data přes wifi. Ten data zaloguje na perzistentní úložiště a pošle je na centrální bránu pomocí wifi sítě, což mi přijde nejednoduší, nemusím nikde tahat kabely, či řešit jiné bezdrátové technologie, poněvadž wifi síť má v místě měření dostatečné pokrytí. Teď se dostávám k takovému kontroverznímu prvku celého flow, a tím je centrální brána. To je v podstatě počítač, který jediný co dělá je přeposílání dat ze senzorů někam jinam, dal by se tedy úplně odstranit s tím že měřící stanice by data odesílala přímo do internetu, ale já ji zahrnul z těchto důvodů. Díky tomu, že s internetem komunikuje pouze jedna stanice nemusím na ostatních řešit jejich autentizaci vůči cloudu, či různé SSL certifikáty a podobně. To mi umožní na jejich místech mohu nasadit mnohem jednoduší zařízení. Dále mi to umožní přístup k datům doma i bez internetu, kdybych si je chtěl nějak zobrazit... A Také to zjednoduší ladění celého systému, kdy například programy mohu testovat u sebe na počítači kdy data budu brát z centrální brány a nebudu muset vůbec zasahovat do kódu v senzoru. No a to je celé z brány pošlu data do cloudu a tam jejich cesta končí, pokud tedy vynechám cestu z cloudu za účelem jejich zobrazení.



Obrázek 2.1: Takto přibližně potečou data

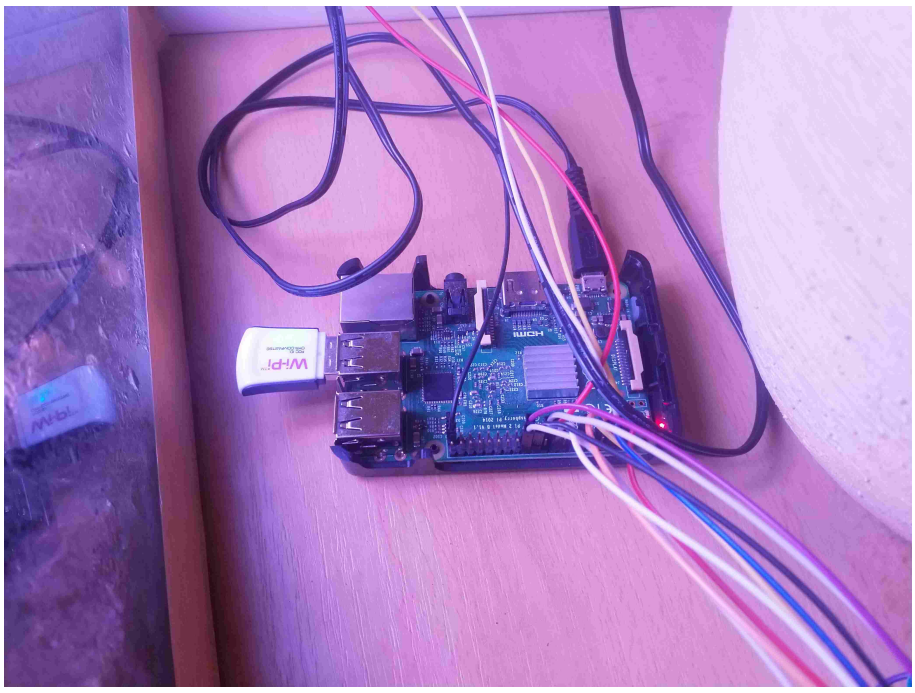
Uživatelské rozhraní pro zobrazení hodnot jsem se rozhodl z důvodu co největší přenositelnosti ji implementovat jako webovou stránku. Takže celá jeho logika a vykreslování bude řešená v javascriptu a až na klientském zařízení, cloud použiji pouze na to, abych z něj vytáhl potřebná data a samozřejmě na hostování celé aplikace. Na této stránce určitě zobrazím graf vývoje změřených hodnot, myslím že v základu by mohlo stačit tak posledních 48 hodin, ale asi bych přidal možnost i delších časových úseků. Z dalších údajů bych si zobrazil tak možná medián naměřených hodnot a možná průměr, ale další údaje mi už přijdou zbytečné. Další analýzy dělat nebudu, spíše bude sledovat jak se kytičkám daří a případně je po nasbírání zkušeností doplním.

Co se otázky bezpečnosti týče, tak vynechám šifrování na cestě od senzoru do brány, zejména z důvodu jednoduššího ladění a implementace, avšak co na této cestě doplním je elektronický podpis zprávy, aby se mi nikdo nepodvrhoval komunikaci. Zabezpečení komunikace s cloudem už budu řešit prostředky té dané služby i co se zobrazení hodnot týče.

Kapitola 3

Hardware

3.1 Měřící stanice



Obrázek 3.1: Raspberry pi 2

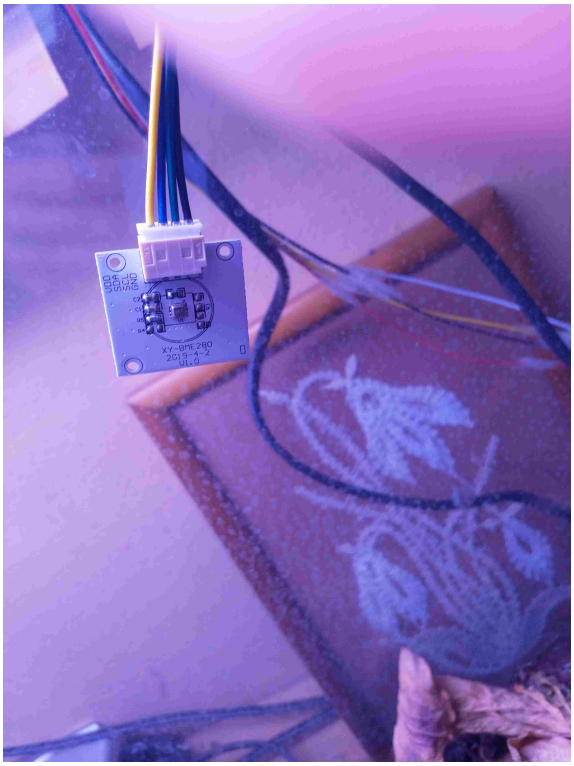
Možných základů pro měřící stanici, jednodeskových počítačů, je dnes na trhu mnoho od různých osmibitů, až po počítače na architektuře ARM, které dosahují výkonu srovnatelného s mobilními telefony. Já jsem pro mé řešení zvolil Raspberry Pi ve verzi 2 a to z několika důvodů. Jednak ho mám k dispozici a dále mi nabízí běžící Linux a tudíž za mne řeší spoustu problémů, od síťové komunikace, po třeba synchronizaci času. Navíc mám k dispozici spoustu digitálních pinů pro připojení různých senzorů, též mám vyřešené i místní úložiště, data se mohou ukládat na SD kartu ze které běží celý systém a výrazně to zjednodušuje vývoj, poněvadž mohu nahrávat nové verze programů vzdáleně, a i vzdáleně sledovat jejich běh, což je pro mě

výhodné, neb terárium nemám v pokoji, kde programuji. Samozřejmě že toto řešení má i své nevýhody. Například v případě, že bych chtěl měřit analogové hodnoty, bych musel dokupovat převodník z analogového signálu na digitální, či v případě nutnosti rozšíření na více míst, by to nebylo ekonomicky výhodné, přeci jen Raspberry Pi stojí kolem 1000 Kč. Nebo pokud bych chtěl zařízení napájet z akumulátoru, tak též s odběrem kolem půl ampéru to nebude to pravé ořechové. Avšak v případě zmíněných problémů mi zvolené celkové řešení umožňuje poměrně komfortně změnit základ stanice na něco vhodnějšího, za předpokladu, že se zvolená deska zvládne připojit na lokální síť. Například mohu použít oblíbenou desku ESP8266 či ESP32, které se cenově pohybují v řádu stokorun, pinů mají dostatek, disponují Wifi čipem a umožňují použití nízko odběrových módů při běhu na baterii.

3.2 Senzory

3.2.1 BME280

Pro měření hodnot v terárium jsem zvolil senzor BME280 od firmy Bosh s cenovkou kolem 100 Kč. Navíc díky sběrnici I2C mi z terária vedou pouze čtyři dráty.



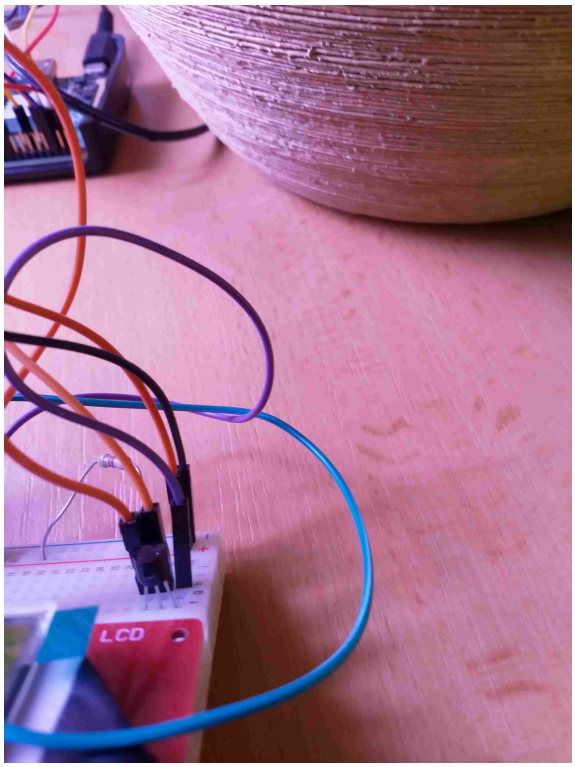
Obrázek 3.2: Modul s BME280, stříbrný čtvereček uprostřed

Teplota	
Rozsah	-40 až +85°C
Rozlišení	0,01°C
Přesnost	± 1°C
Vlhkost	
Rozsah	0 až 100%
Rozlišení	0,008%
Přesnost	±2%
Tlak	
Rozsah	300 až 1100 hPa
Rozlišení	0,18 Pa
Přesnost	1 ± Pa

Tabulka 3.1: Parametry BME280

3.2.2 DS1820

Za účelem případné další analýzy, jsem umístil pár senzorů i mimo terárium, abych mohl sledovat závislost teploty vně a uvnitř.Jako hlavní senzor teploty jsem použil DS1820 s cenou čínské kopie asi 35 Kč.



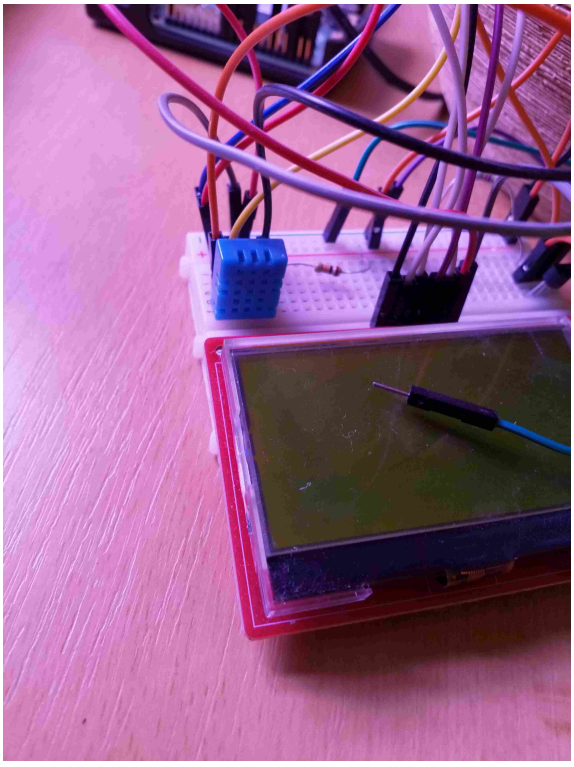
Obrázek 3.3: DS1820

Teplota	
Rozsah	-55 až +125°C
Rozlišení	0,0625°C
Přesnost	± 0,5°C

Tabulka 3.2: Parametry DS1820

3.2.3 DHT11

Pro měření vlhkosti vně terária jsem použil oblíbený senzor teploty a vlhkosti DHT11. Senzor teploty jsem zdvojil z důvodu velké nepřesnosti tohoto modelu.



Obrázek 3.4: DHT11

Teplota	
Rozsah	0 až +50°C
Rozlišení	1°C
Přesnost	± 2°C
Vlhkost	
Rozsah	20 až 90%
Rozlišení	1%
Přesnost	±5%

Tabulka 3.3: Parametry DHT11

3.3 Brána

Jako brána pro komunikaci s internetem se dá taky použit téměř cokoli, ale přeci jen jsou na ní kladeny větší nároky než na měřící stanici. Mimo nutnosti možnosti připojení do sítě je též třeba výpočetní výkon a paměť, dostatečná na komunikaci s cloudem, řešení šifrování, běh nějakého message brokera..., tedy nejlépe nějakou desku s operačním systémem, který mi tohle všechno umožní. Já jsem zvolil opět Raspberry Pi, tentokrát ve verzi 3 opět z velmi jednoduchého důvodu, už mi doma běží, jako takový domácí server.

Kapitola 4

Software

V této kapitole se budu zabývat všemi softwarovými záležitostmi projektu, od volby jazyka, či nějakého frameworku, po detaily implementace. Nebudu však popisovat řádek po řádku, spíše popíši celkové chování a vypíchnu zajímavé části nebo části co mají na chod zásadní vliv, nebo s nimi byl spojen nějaký zajímavý problém. Aktuální verze použitých programů se bude nacházet zde: github.com/prokopparuzek/terarko-program.git.

4.1 Měřící stanice

Jako jazyk pro programování měřící stanice, jsem zvolil Go. Jedna z mnoha výhod je snadná cross-kompilace která mi umožňuje kompilovat programy na svém počítači a do stanice nahrávat už jen binární kód. Čemuž pomáhá i to, že překladač implicitně linkuje staticky.

Obecná koncepce programu je asi takováto. Budu sledovat běh funkce main, kterou jsem se snažil co nejvíce vyčistit.

```
package main

import (
        "os"
        "time"

        "periph.io/x/host/v3"

        stan "github.com/nats-io/stan.go"
        cron "github.com/rk/go-cron"
        log "github.com/sirupsen/logrus"
)

func main() {
        var err error
```

```
        // logrus
        log.SetOutput(os.Stderr)
        log.SetReportCaller(true)
        log.SetLevel(log.ErrorLevel)
        log.SetFormatter(&log.JSONFormatter{})
        f, err := os.OpenFile(logFile, os.O_CREATE|os.
           O_APPEND|os.O_WRONLY, 0664)
        if err != nil {
                log.WithField("file", logFile).Error(err)
        } else {
                log.SetOutput(f)
        }
        forever := make(chan bool)
        for {
                scon, err = stan.Connect("measures", "rpi2",
                     stan.NatsURL("nats://rpi3:4222"), stan.
                    Pings(60, 1440))
                if err != nil {
                        log.Error(err)
                        time.Sleep(time.Second * 30)
                        continue
                }
                break
        }
        defer scon.Close()
        log.Debug("Connected")
        // init periph host
        _, err = host.Init()
        if err != nil {
                log.Fatal(err)
        }
        // Cron
        cron.NewCronJob(cron.ANY, cron.ANY, cron.ANY, cron.
           ANY, 00, 10, sendMeasures)
        cron.NewCronJob(cron.ANY, cron.ANY, cron.ANY, cron.
           ANY, 15, 10, sendMeasures)
        cron.NewCronJob(cron.ANY, cron.ANY, cron.ANY, cron.
           ANY, 30, 10, sendMeasures)
        cron.NewCronJob(cron.ANY, cron.ANY, cron.ANY, cron.
           ANY, 45, 10, sendMeasures)
        //cron.NewCronJob(cron.ANY, cron.ANY, cron.ANY, cron
           .ANY, cron.ANY, 10, sendMeasures)
        log.Debug("Set_CRON")
        <-forever
}
```

Na začátku importuji pár knihoven, ve funkci main, jako první nastavím balíček

logrus, který mi zajišťuje logování, nastavuji co chci logovat, kam a jak to má zformátovat. Poté otevřu spojení na server, inicializuji knihovnu na použití periferií a nastavím cron, aby mi spustil měření každých 15 minut. Nakonec je takový trik, jehož jediným účelem je, aby hlavní gorutina neskončila, jedná se o čtení z kanálu do kterého, však nikdy nic nezapíši. A jelikož jde o blokující operaci, program nikdy neskončí.

Na začátku jsem psal o logování, rád bych ho popsal trochu detailněji. Používám balíček logrus což je taková rozšířená verze balíčku log ze standardní knihovny. Umožňuje mi detailně logovat běh programu a strukturovat logy, například tím, že ke každé zprávě kterou vypíši, mi doplní důležité informace, jako z jaké funkce byl zavolán, na jakém řádku, nebo čas kdy byl zavolán. Toto se dá libovolně měnit. A navíc díky němu mohu nechat ladící hlášky v programu celou dobu, jenom na začátku v inicializaci mu řeknu, že mě zajímají pouze chyby a on mi hlášky s nižší prioritou, tj. debugovací informace a podobně nebude vypisovat. Takže po nasazení, si jen zvolím soubor, kam má logy ukládat, abych o ně nepřišel a prioritu co má vypisovat, a pak můžu jen sledovat chyby.

Samotné získávání dat je velmi jednoduché. Každých 15 minut se zavolá funkce getMeasure(),která postupně projde přes všechny připojené senzory, zavolá funkce, které je obsluhují, v případě chyby je zkouší zavolat vícekrát a vrátí pole s naměřenými hodnotami. Pro měření jsem se snažil co nejvíce využít možností, které poskytuje knihovna periph.io, takže například data vracím ve formě struktury z této knihovny a používám ji pro získávání hodnot ze dvou senzorů, pro třetí ji nepoužívám jen z toho důvodu, že ho zatím nepodporuje. Měření z BME280 je velmi jednoduché. V podstatě otevřu sběrnici, přečtu data a vrátím je, vše za použití výše zmíněné knihovny. Z DS1820 to je velmi podobné, jen nemohu použít periph.io, takže používám knihovnu, jež využívá modul kernelu, který zpřístupňuje tento senzor přes virtuální souborový systém. Senzor DHT11 je na použití asi nejsložitější. Používám sice externí knihovnu, avšak ta pro přístup ke GPIO využívá periph.io. Jinak je měření velmi obdobné jako u ostatních čidel, s tím rozdílem, že je velmi chybové, takže se musí vícekrát opakovat.

DHT11 S tímto senzorem jsem měl asi největší problémy. Nejenom že jsem musel laborovat s nastavením verze api knihovny periph.io, ale i samotná knihovna pro obsluhu senzoru obsahovala chyby. Takže jsem ji důkladně prozkoumal a porovnal s datasheetem k senzoru. Našel jsem dvě zásadní chyby. Jedna byla, že knihovna vyslal příliš krátký startovací pulz, takže ani čidlo neprobudila, to se dalo vyřešit jednoduše, prostě jsem do programu přidal pauzu. A druhá, že špatně zpracovávala data co z čidla četla. Zpracovávala je jako jedno velké šestnáctibitové číslo, ale v datasheetu jsem se dočetl že tyto dva bajty představují číslo v desetinné čárce, ale velmi zvláštně. První bajt je celá část a druhý desetinná a zároveň s tím se v datasheetu píše že rozlišovací schopnost senzoru je 1 °C a 5 %, takže jsem druhý bajt zahodil a dál pracoval pouze s tím prvním. A to fungovalo na jedničku. Tuto zvláštnost s rozlišením bych asi vysvětlil tím, že komunikační protokol bude schodný i s dražšími senzory s lepším rozlišením a to možná souvisí i s chybami v knihovně, poněvadž je určena i pro ně. Takže abych jejich podporu nerozbil jsem mé úpravy

podmínil použitím správného typu čidla. Moje upravená verze kterou používám, je k nalezení zde: github.com/prokopparuzek/go-dht.git

Pro ukládání dat na měřící stanici jsem nevymýšlel nic složitého. Vezmu jen data z každého senzoru, přidám timestamp, počet milisekund od 1.1. 1970, tím se vyhnu problémům s reprezentací času a převody časových pásem a to vše přidám za konec souboru konkrétního senzoru. Data ukládám ve formátu CSV, tedy oddělené čárkou. Jednotlivé hodnoty/sloupce nijak neoznačuji, nechávám to na utilitách, jejichž cílem bude data obnovit, ty jediné s nimi takto budou pracovat a to jen občas, takže to myslím nevadí, a navíc to zjednodušuje kód.

NATS Zde bych rád zmínil něco o message brokeru NATS, který používám. Prvně co to vlastně znamená ten message broker? Jedná se o server zajišťující komunikaci, obvykle se používá právě v IoT nebo třeba v distribuovaných systémech. Funguje na principu tzv. témat asi takto, já mu pošlu zprávu s určitým tématem (názvem), obsahující zrovna třeba naměřené teploty a on ji přepošle všem, kdo jsou přihlášeni k odběru zpráv daného tématu. Může se stát, že se jednotlivé části v různých programech jmenují jinak, ale princip je stejný. Toto je takzvaná Publish-Subscribe strategie, její vlastnost však je, že server zprávu pošle a pak ji zahodí, takže pokud někomu nepřijde, třeba z důvodu výpadku proudu... tak už ji nikdy nedostane. Někomu to může vadit, takže pak vznikají nadstavby, kdy server zprávu uloží a zkouší ji poslat, dokud od klienta neobdrží potvrzení o přijetí, to je mnohem robustnější řešení,a proto ho použiji i já, konkrétně použiji nadstavbu nad serverem NATS nazývanou NATS-streaming. Stejně jako NATS se jedná o lehkou aplikaci napsanou v go, takže zabírá minimum zdrojů. Já jsem ji vybral z důvodu jednoduchosti, dostupnosti široké škály klientských knihoven a též již zmiňované lehkosti. A taky proto že mám rád go.

Odesílání dat začíná vlastně už na úplném začátku, kdy se spojím se serverem a pak až do konce držím spojení otevřené. Samotné odeslání dat do message brokera, se vlastně moc neliší od uložení. Taky vezmu data, přidám timestamp a pošlu je. Je tu však pár rozdílů. Data posílám ve formátu JSON, který je jednoduchý a čitelný. Z důvodu možných latencí, nedostupnosti sítě... odesílám každou zprávu v samostatné gorutině, abych neblokoval další měření, jelikož když se zprávu nepovede odeslat, tak ji zkouším poslat po minutě další zhruba dvě hodiny. No a to je vše po odeslání zprávy už nemusím nic řešit, poněvadž server si ji uloží a pošle dál, takže už se neztratí.

4.2 Domácí gateway

4.3 Cloud

4.4 Zobrazení grafů

Kapitola 5

Výsledek

Závěr

Literatura

LOURME, Olivier, 2018a. *Post 1 of 3. Our IoT journey through ESP8266, Firebase and Plotly.js* [online] [cit. 2020-11-15]. Dostupné z: https://medium.com/@o.lourme/our-iot-journey-through-esp8266-firebase-angular-and-plotly-js-part-1-a07db495ac5f.

LOURME, Olivier, 2018b. *Post 2 of 3. Our IoT journey through ESP8266, Firebase and Plotly.js* [online] [cit. 2020-11-15]. Dostupné z: https://medium.com/@o.lourme/our-iot-journey-through-esp8266-firebase-angular-and-plotly-js-part-2-14b0609d3f5e.

LOURME, Olivier, 2018c. *Post 3 of 3. Our IoT journey through ESP8266, Firebase and Plotly.js* [online] [cit. 2020-11-15]. Dostupné z: https://medium.com/@o.lourme/our-iot-journey-through-esp8266-firebase-angular-and-plotly-js-part-3-644048e90ca4.

TIŠŇOVSKÝ, Pavel, 2019a. *Komunikace s message brokery z programovacího jazyka Go* [online] [cit. 2020-11-15]. Dostupné z: https://www.root.cz/clanky/komunikace-s-message-brokery-z-programovaciho-jazyka-go/.

TIŠŇOVSKÝ, Pavel, 2019b. *NATS Streaming Server* [online] [cit. 2020-11-15]. Dostupné z: https://www.root.cz/clanky/nats-streaming-server/.

TIŠŇOVSKÝ, Pavel, 2019c. *Použití message brokeru NATS* [online] [cit. 2020-11-15]. Dostupné z: https://www.root.cz/clanky/pouziti-message-brokeru-nats/.

TIŠŇOVSKÝ, Pavel, 2020. *Tvorba grafů v jazyce Go: kreslení ve webovém klientu* [online] [cit. 2020-11-15]. Dostupné z: https://www.root.cz/clanky/tvorba-grafu-v-jazyce-go-kresleni-ve-webovem-klientu/.

Seznam obrázků

Seznam tabulek

Slovníček pojmů

cross-kompilace Překlad programu pro jinou platformu, než na které je překládán. 19

garbage collector Způsob automatické správy paměti. Funguje tak, že speciální algoritmus (garbage collector) vyhledává a uvolňuje úseky paměti, které již program nebo proces nepoužívá. Programátor to tedy již nemusí řešit a tím odpadá celá řada chyb způsobených zapomenutím na uvolnění paměti... Nevýhodou je, že si garbage collector ukousne část výkonu procesoru pro sebe, takže pak program běží pomaleji. 29

Go Kompilovaný, staticky typovaný jazyk od Googlu, a na jehož vývoji se podílel například Ken Thompson, spolutvůrce programovacího jazyka C, z jehož syntaxe vychází i syntaxe Go. Cíle jazyka jsou zejména jednoduchá syntax, strmá křivka učení, či snadná tvorba vícevláknových aplikací. Kompromisem v návrhu jazyka bylo zahrnutí garbage collector, programy jsou sice pomalejší, ale kód jednodušší. Jazyk je oblíben i mimo Google, je v něm napsán například Docker, či ho používá Dropbox. 19

linkování Proces spojení objektového souboru vygenerovaného překladačem s knihovnami, či jinými soubory. Existují dva typy dynamické, kdy se knihovny připojují až za běhu a jsou společné pro všechny programy běžící na daném počítači a statické, kdy jsou všechny knihovny přibaleny k výslednému spustitelnému souboru. 19

Příloha A

Zdrojový kód