

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 mým rodičům za pomoc s debugováním mé seminární práce.

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 nové senzory nebo místo.

Klíčová slova: měření, IoT, terárium

Annotation: The goal is a creation of automatic system to monitor temperature, humidity and another quantity in my terrarium with carnivorous plant. Also I want to put measurements online in a form of graph so user can watch it in real time. Also I want the system to be easily extendable with new sensors or rooms.

Key words: measure, IoT, terrarium

Obsah

Úvod	9
1 Požadavky na řešení	11
2 Analýza problému	13
3 Hardware	15
3.1 Měřící stanice	15
3.2 Senzory	16
3.2.1 BME280	16
3.2.2 DS18B20	17
3.2.3 DHT11	18
3.3 Brána	19
4 Software	21
4.1 Měřící stanice	21
4.2 Cloud	23
4.2.1 Firebase	23
4.2.2 Firestore	26
4.2.3 Hosting	28
4.2.4 Webová aplikace	28
4.3 Domácí gateway	31
4.4 Zobrazení grafů	33
Závěr	37
Literatura	39
Seznam obrázků	42
Seznam tabulek	43
Slovníček pojmu	45
Akronypy	49
A Zdrojový kód	51
A.1 Měřící stanice	51
A.1.1 global.go	51
A.1.2 measure.go	52

A.1.3	getData.go	53
A.1.4	csvSave.go	55
A.1.5	sendData.go	56
A.2	Brána	58
A.2.1	fire.go	58
A.3	Webová aplikace	60
A.3.1	index.html	60
A.3.2	plot.js	61



Úvod

Už potřetí zahajuji svůj pokus pěstovat masožravé rostliny, který zatím vždy skončil jejich úhynem. Z tohoto důvodu jsem se rozhodl začít sledovat prostředí v teráriu, kde je pěstuji, abych mohl v případě úhynu určit z jakého důvodu uhynuly. Přehráli se, umrzli, uschly... většinou z důvodu mé nepřítomnosti, kdy jsem je nemohl kontrolovat. Avšak mnohem radší 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, která 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, zamezení neoprávněného přístupu. Z důvodů urychlení a zlevnění vývoje nebudu vždy používat nejhodnější, 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 propsalala do cloudu, tedy i v případě nějakého výpadku se, poté odeslala data co jsem změřil, ale neodeslal. Když přijdu o jedno měření během krátkého výpadku, tak to vadit nebude.

Základní požadavek je pouze zobrazení dat. 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, nevyšší, nejnižší hodnota či medián...

Nároky na uživatelské rozhraní v podstatě nemám. Pro správu senzorů je nepotřebné. Stejně moc obměňovat či přidávat nebudu, a i kdyby stejně si pro ně budu muset napsat obslužný program. Udělat nějakou knihovnu pro více senzorů není mým cílem. Pro zobrazení hodnot je důležité, ale ohledně nároků mi bude stačit velmi jednoduché rozhraní, stačit bude jediná stránka. 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 vyneschal.

K bezpečnosti bych rád zmínil toto. Bylo by dobré mít komunikaci po celé 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, bych do požadavků zahrnul. 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 kyticíky, ale mohlo by se z toho dát odvodit, že je nezalévám tj. jsem pryč a v bytě nikdo není. Pokud si někdo dá tu práci, že mi napichne připojení, nebo přijede a odposlechné



data z domácí sítě, tak má asi i možnosti jak si to, že nejsem doma zjistit jinak. Rozhodně to ale bude složitější než otevření webové stránky.

Kapitola 2

Analýza problému

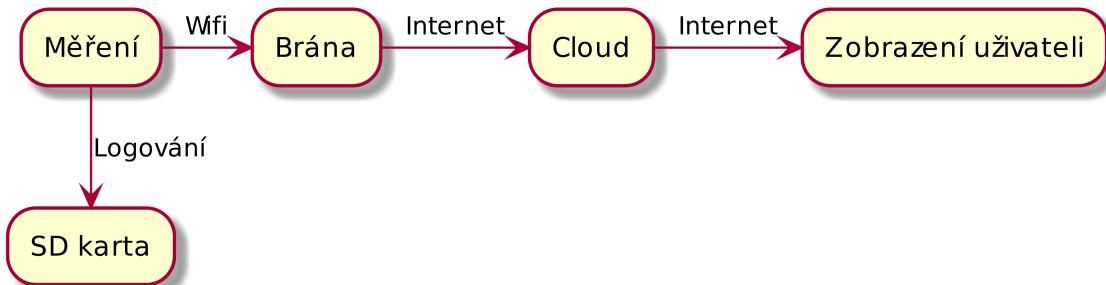
Ohledně měření bych zatím zmínil jen frekvence, o které si myslím, že pro mé účely bude stačit měřit jednou za čtvrt hodiny, to je 96 měření za den. Neodchytím tím sice drobné výkyvy v rámci minut, ale cílem je zachytit dlouhodobý průběh hodnot v rámci dne, kdy mne tak drobné výkyvy nezajímají.

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. 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.

Ted' se dostávám k samotnému data flow, tedy jak a kam mi potečou data, co naměřím. Začnu u senzoru, ten změří data a poše 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 poše je na centrální bránu pomocí wifi sítě, což mi přijde nejjednodušší. Nemusím nikde tahat kabely či řešit jiné bezdrátové technologie, poněvadž wifi síť má v místě měření dostatečné pokrytí. Ted' se dostávám k takovému kontroverznímu prvku celého flow a tím je centrální brána. Ta by se dala by se tedy úplně odstranit s tím, že měřící stanice by data odesíala 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 nasadit mnohem jednodušší 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

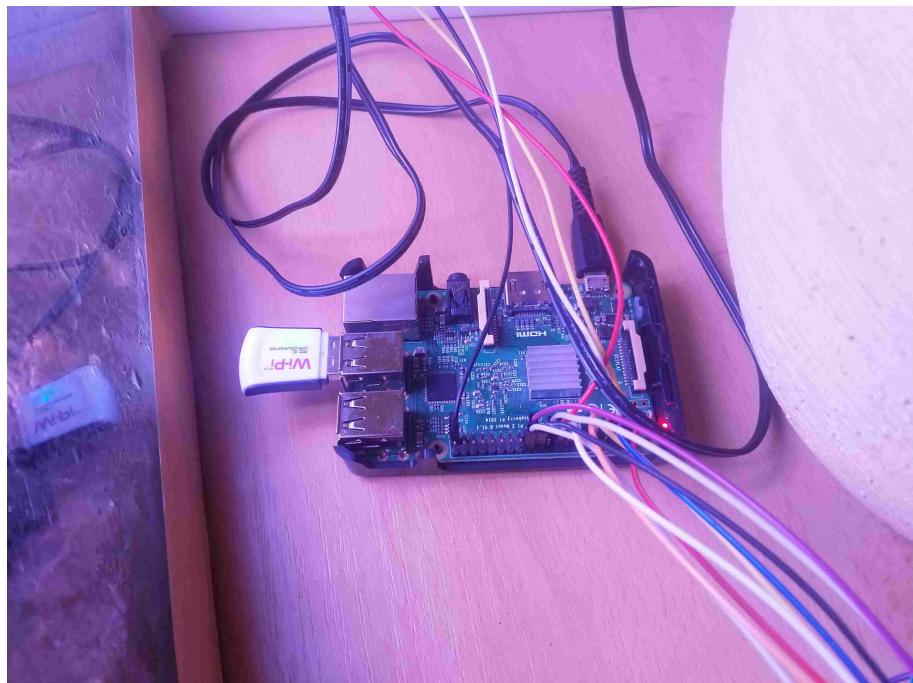
Uživatelské rozhraní pro zobrazení hodnot jsem se rozhodl z důvodu co největší přenositelnosti implementovat jako webovou stránku. Takže celá jeho logika a vykreslování bude řešená v javascriptu a až na klientském zařízení. Cloud použijí pouze na to, abych z něj vytáhl potřebná data a samozřejmě na hostování celé aplikace. Na této stránce určitě zobrazím graf vývoje změřených hodnot, za posledních 48 hodin, ale asi bych přidal možnost zobrazení i delších časových úseků.

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 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

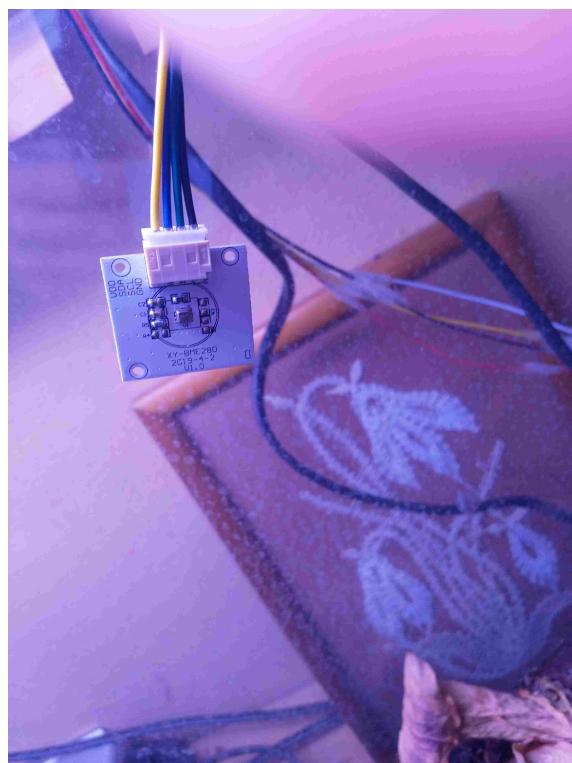
Možných základů pro měřící stanici, je dnes na trhu mnoho, od různých osmiček až po počítače na architektuře ARM, které dosahují výkonu srovnatelného s mobilními telefony (Wikipedie, 2021e). Já jsem pro mé řešení zvolil Raspberry Pi ve verzi 2 a to z několika důvodů. Jednak ho mám k dispozici a dále mi nabízí běžící Linux a tudíž za mne řeší spoustu problémů, od síťové komunikace po 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. 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. 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 s odběrem kolem půl ampéru by moc dlouho nevydržel (Wikipedie, 2021e). 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 (Stehlík, 2019).

3.2 Senzory

3.2.1 BME280

Pro měření hodnot v teráriu jsem zvolil senzor BME280 teploty vlhkosti a tlaku vzduchu od firmy Bosch s cenovkou kolem 200Kč. Navíc díky sběrnici I²C mi z terária vedou pouze čtyři dráty (Pájeníčko, 2020b).



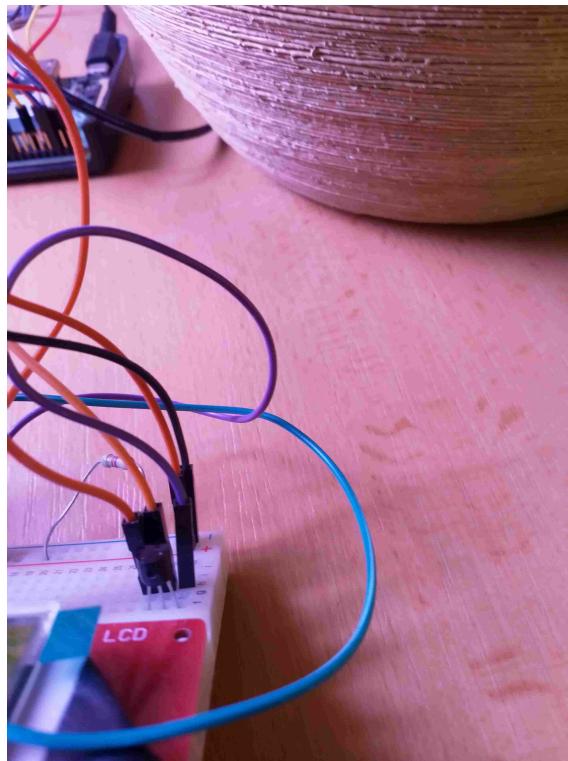
Obrázek 3.2: Modul s BME280, stříbrný čtvereček uprostřed

Teplota	
Rozsah	-40 až +85°C
Rozlišení	0,01°C
Přesnost	± 1°C
Vlhkost	
Rozsah	0 až 100%
Rozlišení	0,008%
Přesnost	±3%
Tlak	
Rozsah	300 až 1100 hPa
Rozlišení	0,18 Pa
Přesnost	1 ± Pa

Tabulka 3.1: Parametry BME280

3.2.2 DS18B20

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 DS18B20 vyvinutý firmou Dallas s cenou čínské kopie asi 35 Kč. Jde o můj oblíbený senzor, poněvadž je přesný, snadno použitelný a díky sběrnici OneWire mu stačí pouze tři, případně dva dráty (Pájeníčko, 2020a).



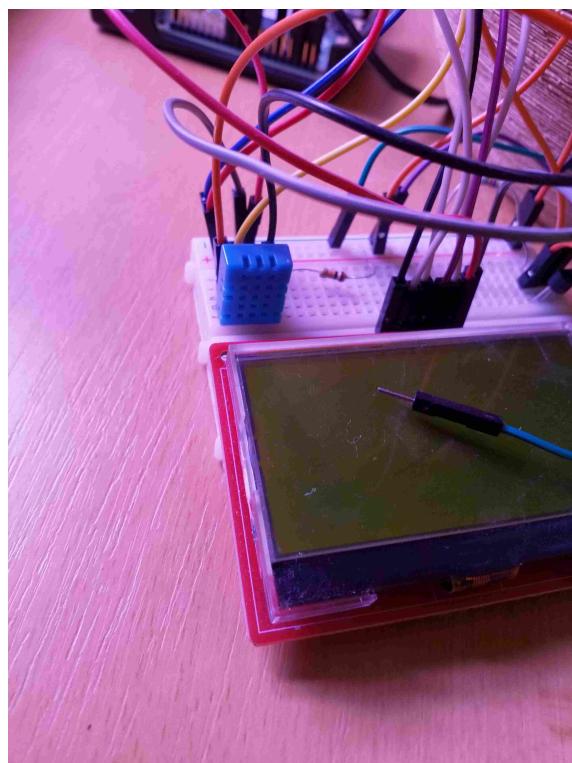
Obrázek 3.3: DS18B20, pouzdro TO-92

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

Tabulka 3.2: Parametry DS18B20

3.2.3 DHT11

Pro měření vlhkosti vně terária jsem použil oblíbený senzor teploty a vlhkosti DHT11 s cenovkou kolem 40 Kč. Senzor teploty jsem zdvojil z důvodu velké nepřesnosti tohoto modelu. S tímto senzorem jsem měl největší problémy, zejména díky jeho nestandardní sběrnici, která připomíná OneWire, ale používá jiný komunikační protokol (Pájeníčko, 2020c).



Obrázek 3.4: DHT11, modrý obdélníček

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 použít téměř cokoli, ale přeci jen jsou na ní kladený 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

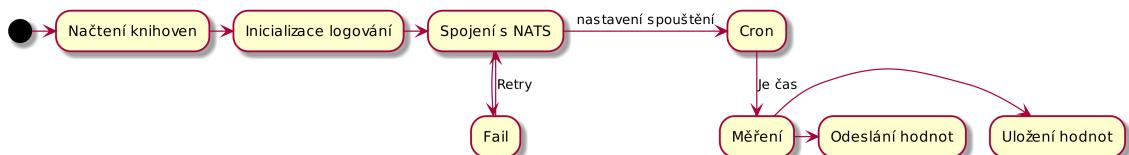
Software

V této kapitole se budu zabývat všemi softwarovými záležitostmi projektu od volby jazyka 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: <https://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-komplikace, 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.

Průběh programu, který provádí měření, vypadá takto.



Obrázek 4.1: Průběh programu

Na začátku importuji šest knihoven, konkrétně používám části standardní knihovny pro práci s časem a operačním systémem, knihovnu pro práci s periferiemi a dále to jsou knihovny pro spojení s bránou, umožnění automatického spouštění programů v daný čas a logování. Ve funkci main jako první nastavím knihovnu logrus, která mi zajišťuje logování, nastavuji co chci logovat, kam to má zapsat a jak to má zformátovat. Poté otevřu spojení s bránou, 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 nezapíší. A jelikož jde o blokující operaci, program nikdy neskončí.

Měření Samotné získávání dat je velmi jednoduché. Každých 15 minut Cron zavolá funkci `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 ne-používám 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 DS18B20 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 (<https://www.mouser.com/datasheet/2/758/DHT11-Technical-Datasheet-Translated-Version-1143054.pdf>) k senzoru. Našel jsem dvě zásadní chyby. Jedna byla, že knihovna vyslala 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. Před úpravou mi vracela nereálné hodnoty teploty a vlhkosti, konkrétně vracela chybu, že načtená data jsou mimo rozsah, protože hodnoty zpracovávala jako jedno velké šestnáctibitové číslo, ale v datasheetu jsem se dočetl že tyto dva bajty představují číslo v desetinné čárce, ale velmi neobvykle. První bajt je celá část a druhý desetinná a zároveň s tím se v datasheetu píše, že rozlišovací schopnost senzoru je 1 °C a 1 %, 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 shodný 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: <https://github.com/prokopparuzek/go-dht.git>.

Ukládání Pro ukládání dat na měřící stanici jsem nevymýšlel nic složitého. Vezmu jen data z každého senzoru, přidám timestamp, 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. Jednotlivé hodnoty/sloupce nijak neoznačuju, 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 zjednoduší kód.

Message broker Jako základní message broker by se dal použít server NATS. Já použiji trochu bezpečnější variantu, konkrétně použiji nadstavbu nazývanou NATS-streaming. Stejně jako NATS se jedná o lehkou aplikaci napsanou v Go, takže zabírá minimum zdrojů (Tišňovský, 2019b). Já jsem ji vybral z důvodu jednoduchosti, dostupnosti široké škály klientských knihoven a též již zmiňované lehkosti.

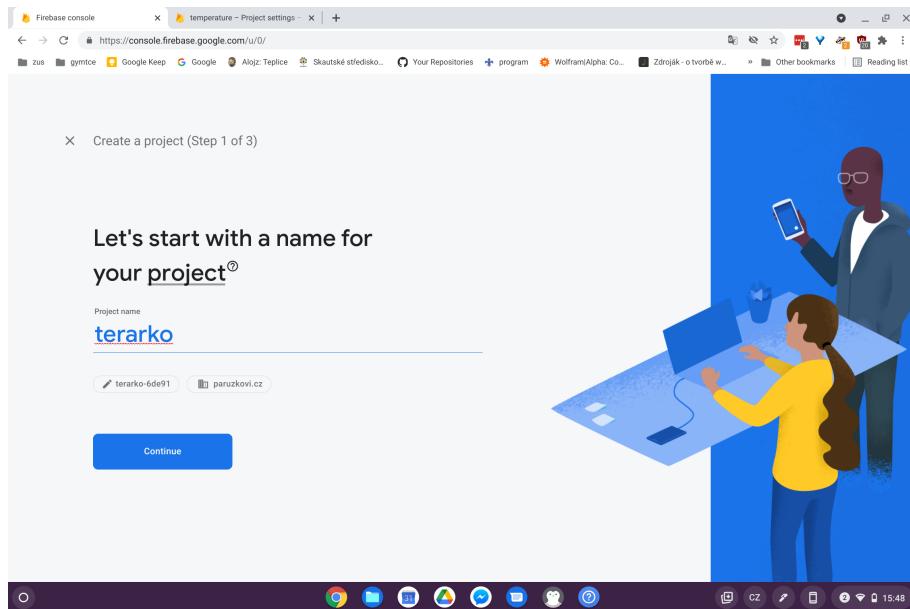
Odesílání Odesílání dat začíná vlastně už na úplném začátku, kdy se spojím se bránou a pak až do konce držím spojení otevřené. Samotné odeslání dat do message brokera se vlastně moc neliší od uložení. Taky vezmu data, přidám timestamp a pošlu je. Je tu však pár rozdílů. Data posílám ve formátu JSON, 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 Cloud

Na internetu se dá najít spoustu cloudových řešení, které bych mohl využít. Řešení z nichž některá byla přímo vytvořená pro sběr a zobrazení dat z chytrých senzorů, se mi úplně nelíbila. Důvodů bylo několik. Řešení vytvořená „na míru“ měla zásadní problém v tom, že většinou v tarifu zdarma měla nějaké významné omezení, například omezení ukládaných dat pouze na poslední tři měsíce a platit se mi nechtělo(<https://io.adafruit.com/>). Z ostatních jsem vybral Firebase, poněvadž měl rozumně nastavené limity zdarma, nádhernou dokumentaci a zároveň za ním stojí velká firma, takže se nemusím bát, že za rok skončí(<https://firebase.google.com/pricing>).

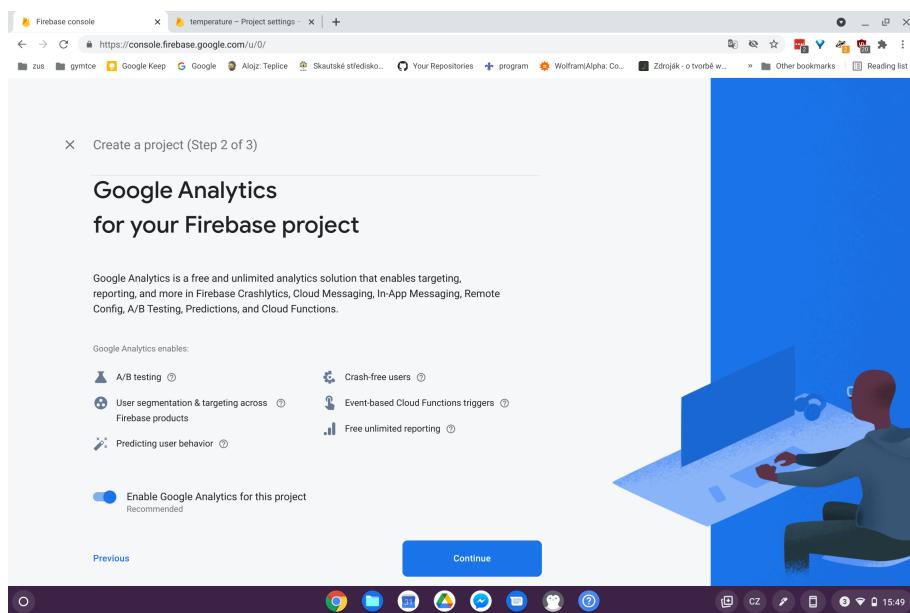
4.2.1 Firebase

Firebase je služba Googlu, takže jediné co je potřeba pro založení je Google účet, díky tarifu zdarma ani nechce žádnou platební kartu. Založení projektu se pokusím osvětlit několika obrázky. Počáteční adresa je <https://console.firebaseio.google.com/u/0/>



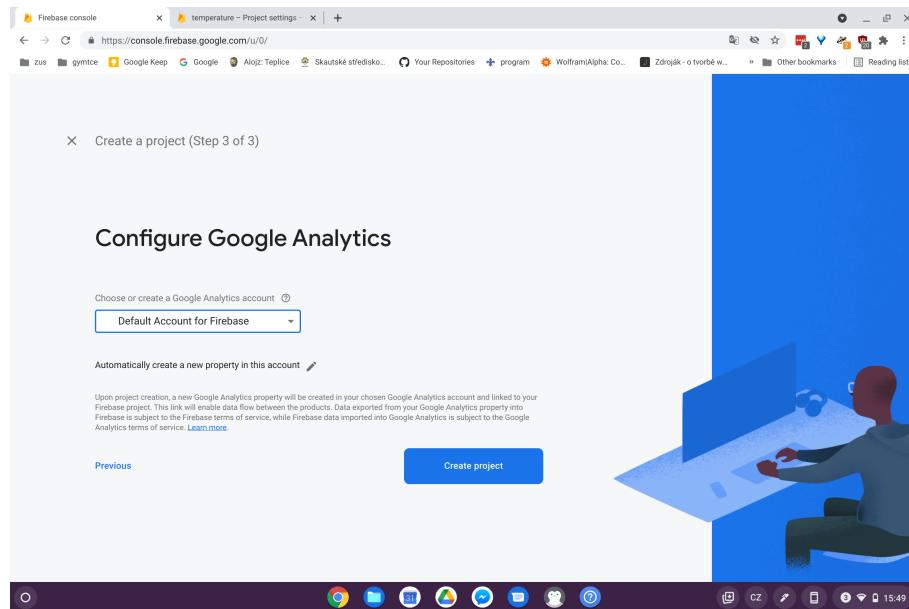
Obrázek 4.2: Firebase, krok 1

V kroku 1 vybírám projektu jméno.



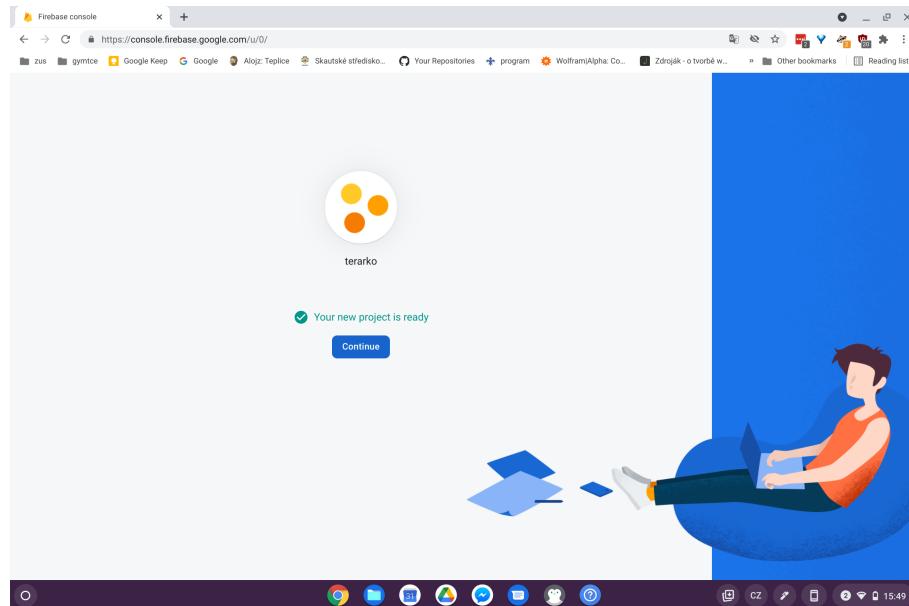
Obrázek 4.3: Firebase, krok 2

V kroku 2 můžu pro projekt povolit Google Analytics.



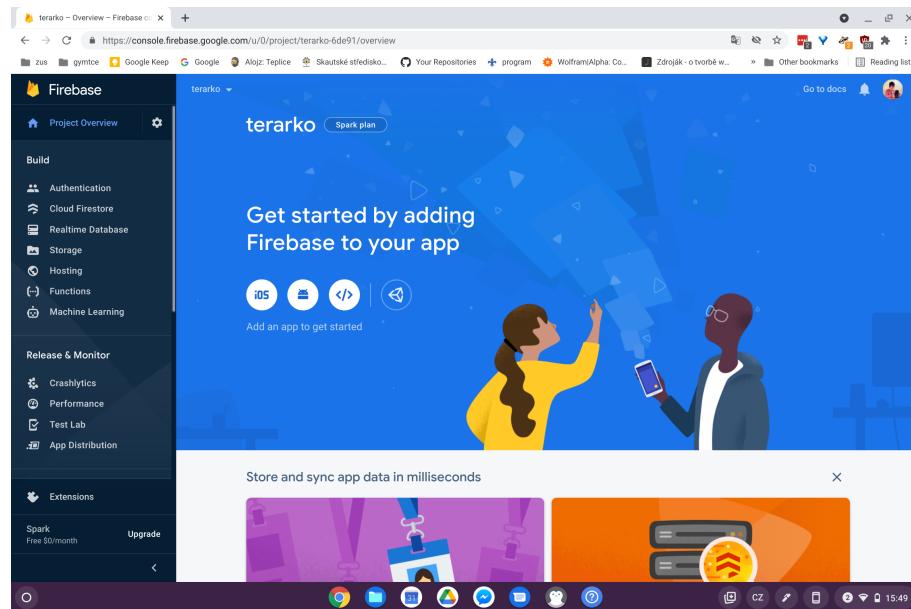
Obrázek 4.4: Firebase, krok 3

V kroku 3 přiřazují účet Google Analytics.



Obrázek 4.5: Firebase, hotovo

Vše jsem ponechal na výchozích hodnotách, teoreticky bych vzhledem k účelu mohl vypnout Google Analytics.

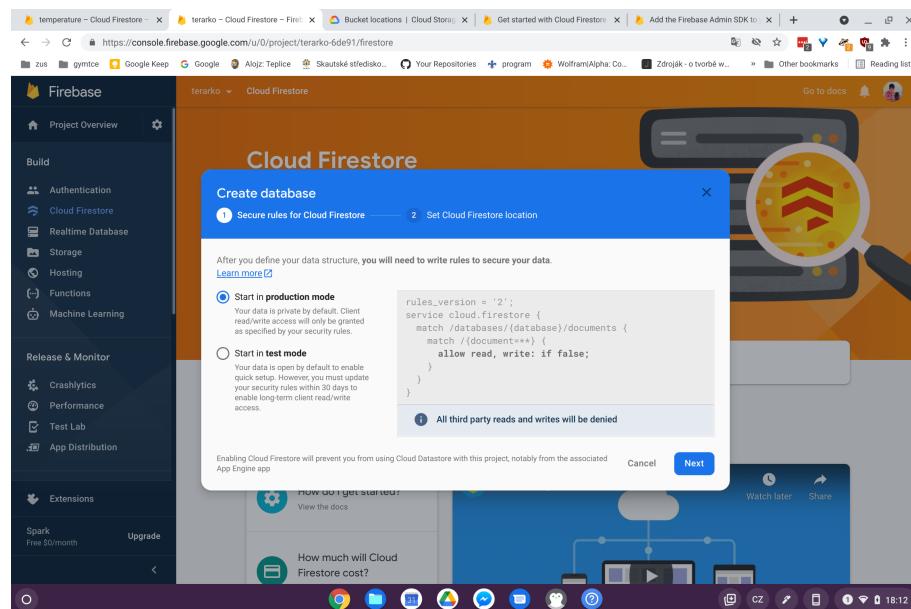


Obrázek 4.6: Firebase

Takto vypadá úvodní stránka Firebase po založení. Teď můžu přidat aplikace, které budu používat.

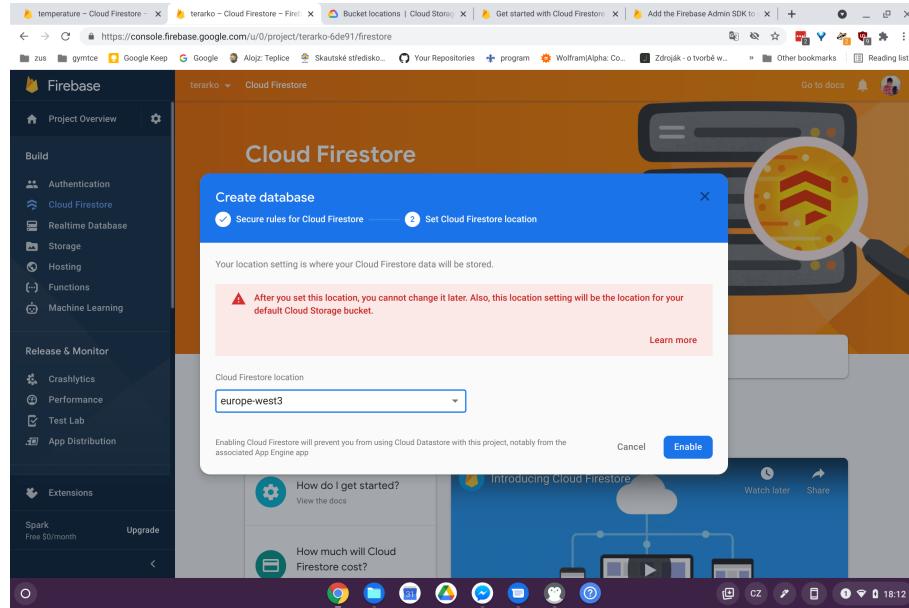
4.2.2 Firestore

Jako první přidám firestore, což je rychlá NoSQL databáze, kterou budu používat pro ukládání naměřených hodnot. Opět se pokusím vysvětlit několika obrázků.



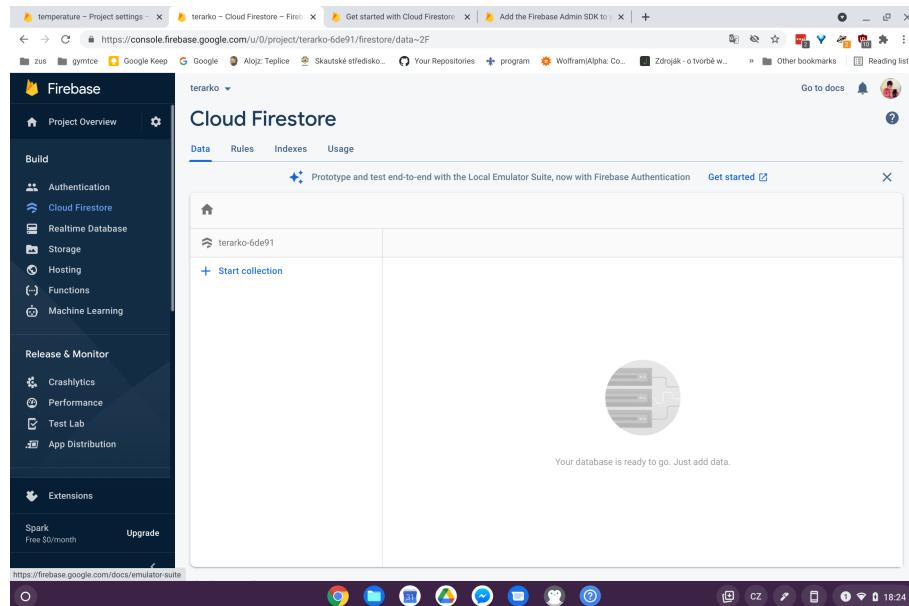
Obrázek 4.7: Firestore, krok 1

Nastavení bezpečnosti použiji v produkčním módu, nechci aby se mi tam někdo připojoval bez přihlášení.



Obrázek 4.8: Firestore, krok 2

Jako lokaci databáze volím Evropu konkrétně Frankfurt.



Obrázek 4.9: Firestore

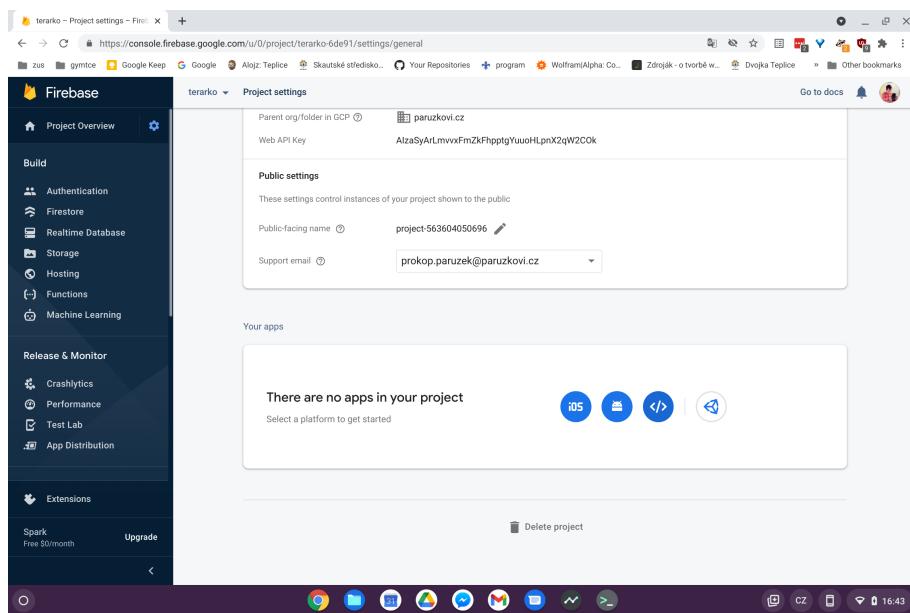
A mám nastaveno. Teď můžu začít přidávat data.

4.2.3 Hosting

Pro hosting jsem použil též platformu Firebase. Při založení jsem postupoval podle návodu (<https://firebase.google.com/docs/hosting>). Během inicializace jsem zvolil, že chci pouze hosting, pro projekt terárko a povolil jsem GitHub Actions. Takže po každém commitu, který pošlu na GitHub se mi automaticky nasadí nejnovější verze stránky. Kvůli tomu jsem zdrojové kódy umístil do samostatného repozitáře.

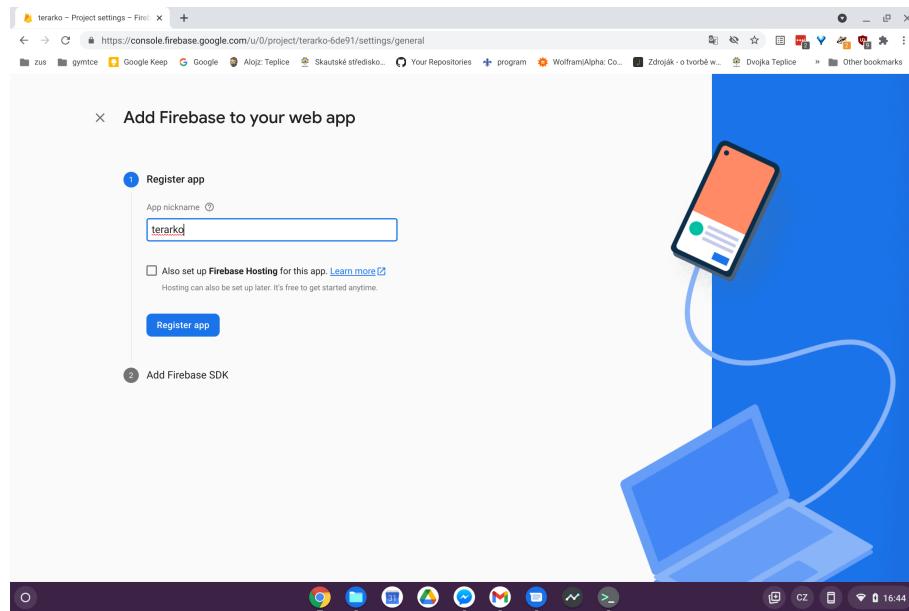
4.2.4 Webová aplikace

Poslední co je třeba nastavit je samotná webová aplikace a její propojení s hostingem. Opět znázorním několika obrázky.

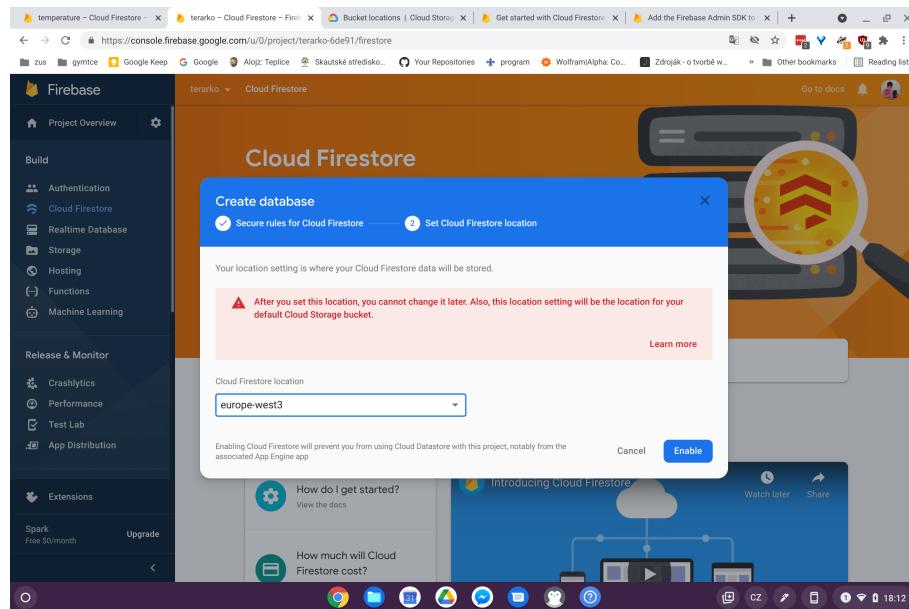


Obrázek 4.10: Přidání aplikace

Přidávám webovou aplikaci, tj. špičaté závorky s lomítkem.



Obrázek 4.11: Jméno aplikace



Obrázek 4.12: Firestore SDK

Toto je důležité pokud si aplikaci hostujete někde u sebe. Já použiji Firebase hosting a tudiž mi to stačí odkliknout.



The screenshot shows the 'Project settings' page for a project named 'terarko'. On the left sidebar, under 'Web apps', there is a configuration for a 'Web App' with the name 'terarko'. The 'SDK setup and configuration' section contains a script tag for the Firebase JS SDK. The script includes the app ID, auth domain, project ID, storage bucket, messaging sender ID, and an app ID. The configuration is set to 'Config'.

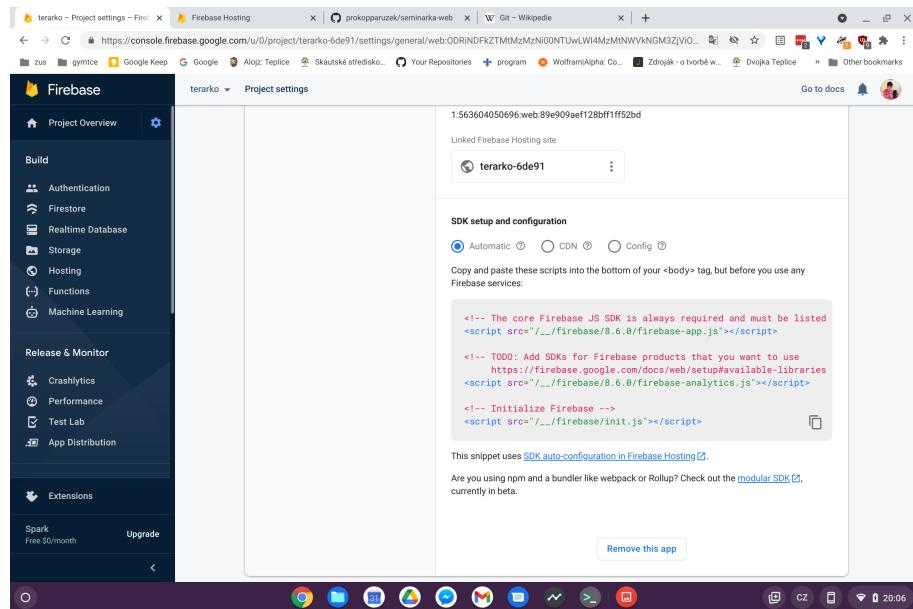
Obrázek 4.13: Propojení s hostingem

Nyní chci propojit aplikaci s firebase hostingu.

A modal dialog box titled 'Link to a Firebase Hosting site' is displayed. It contains instructions for linking a web app with Firebase Hosting. A dropdown menu is open, showing the option 'terarko-6de91'. Below the dropdown, there is a 'Link' button.

Obrázek 4.14: Vybrání hostingu

Zde vyberu, jaký hosting chci použít, a to mi umožní zjednodušit kód stránek, protože si všechny ty konfigurační údaje načte přímo z hostingu spolu s Firebase knihovnami.

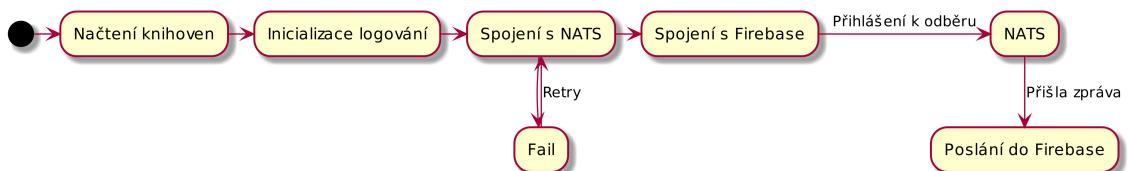


Obrázek 4.15: Zjednodušený kód

Zde je vidět, jak z mnoha řádků kódu zbyly pouze tři při využití Firebase hostingu.

4.3 Domácí gateway

Domácí brána je v podstatě velmi jednoduchý program, který pouze přeposílá data, jež obdrží od NATS serveru, do Firebase Firestoru. Z důvodů popsaných výše jsem jako jazyk opět zvolil Go.

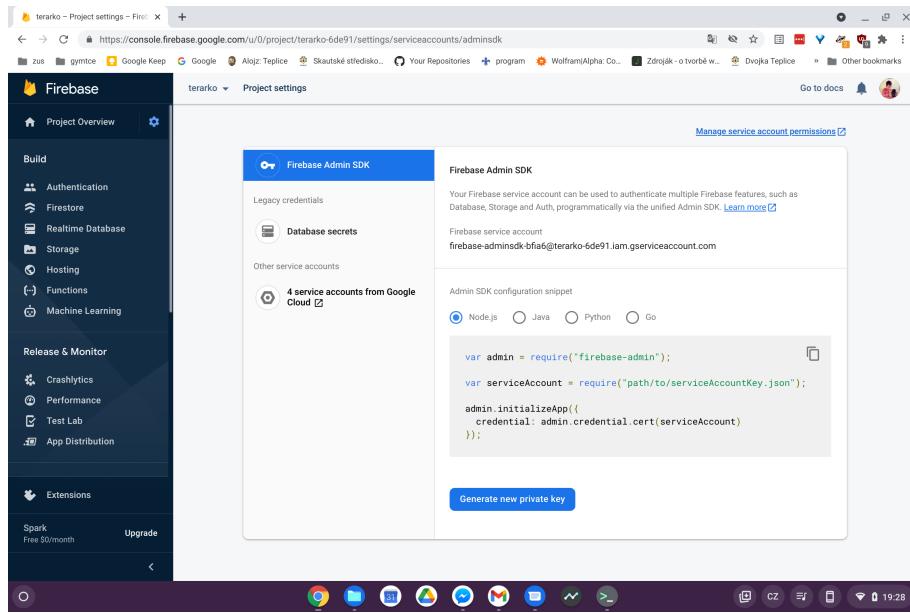


Obrázek 4.16: Průběh programu

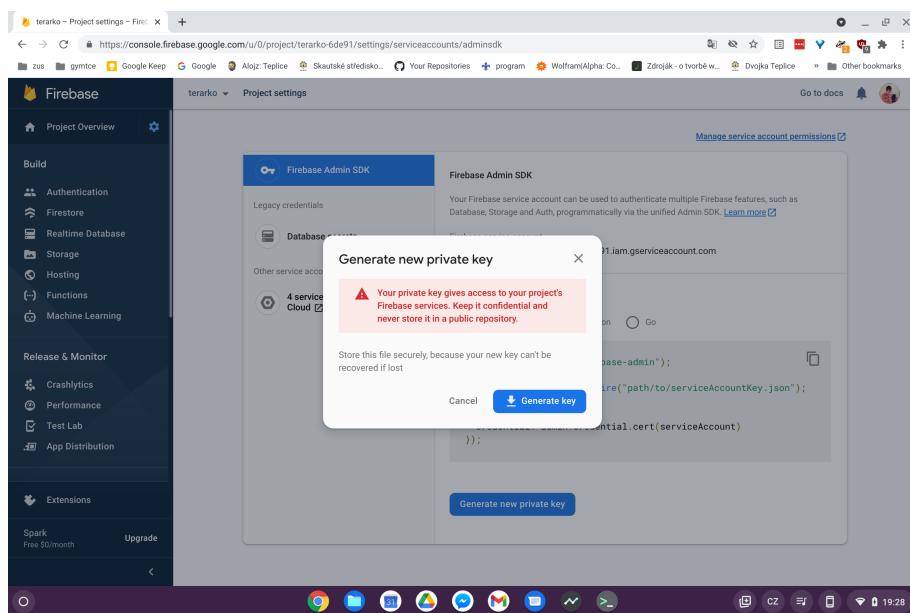
Spojení s NATSem je stejné jako u odesílání dat. Jediný rozdíl je v aplikační části, kdy data neodesílám, ale přijímám. Příjem probíhá tak, že se přihlásím k odběru kanálu, do kterého měřící stanice odesílá data a pro každou zprávu, která přijde spustím funkci, která ji pošle dál. Trochu nestandardní je přihlášení k odběru. Používám takzvanou „durable subscription“. Jedná se o funkci NATS streaming serveru, která mi umožňuje přihlásit se pod určitým jménem a server si pak pamatuje, jakou poslední zprávu mi posílal. Takže když dojde k výpadku, po opětovném připojení mi pošle všechny zprávy, které jsem dosud nedostal (Tišňovský, 2019b).

Jediné co funkce zpracovávající naměřená data dělá, je přeposílání do cloudu. Celou komunikaci s cloudem jsem v podstatě opsal z dokumentace firestoru (<https://>

//firebase.google.com/docs/firestore). Začíná to inicializací spojení. Přihlašování probíhá přes vygenerovaný soubor s privátním klíčem. Soubor jsem pouze stáhnul, uložil někam, kde k němu má aplikace přístup, a řekl jsem jí, kde ho má hledat. O zbytek se postará Googlem poskytovaná knihovna. To proběhne po startu programu, a pak se vytvořené spojení používá až do konce. Poté už se jen přeposílájí data.



Obrázek 4.17: Vygenerování klíče, krok 1

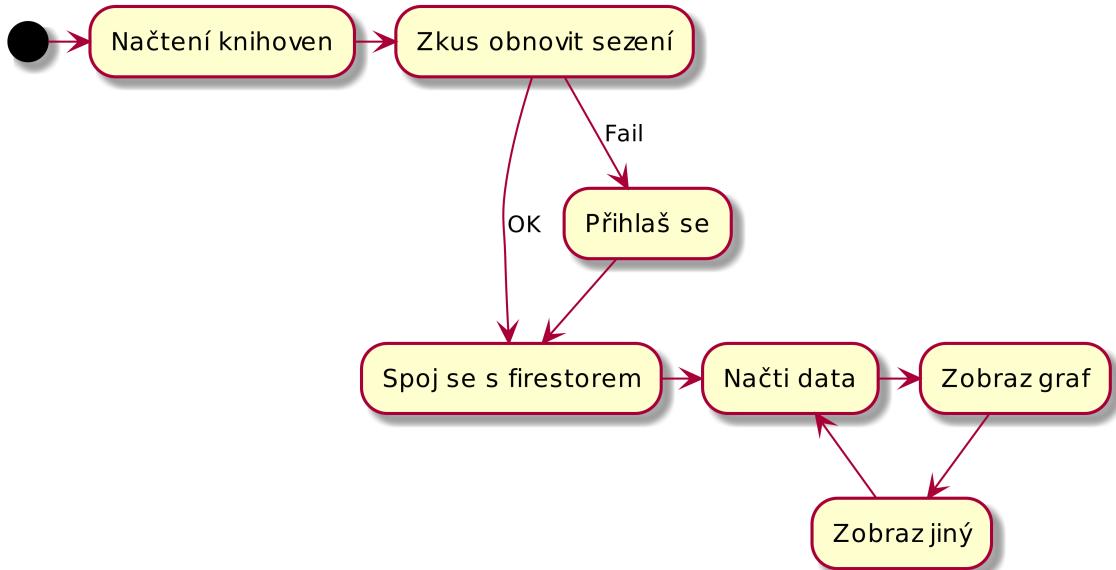


Obrázek 4.18: Vygenerování klíče, krok 2

Organizace dat Pro každý senzor mám v databázi samostatnou kolekci, ve které je jedno měření představováno souborem, jenž obsahuje naměřená data a timestamp. Název souboru nechávám generovat firestore, pro mě je nepodstatný.

4.4 Zobrazení grafů

Pro potřeby zobrazení jsem se rozhodl vytvořit velmi jednoduchou Single-page application. Jediné co se na ní bude nacházet budou tři tlačítka pro přepínání mezi jednotlivými senzory a místo pro graf hodnot. Celá aplikace bude napsaná v javascriptu. Nejsem v něm úplně zběhlý, takže je možné, že některé konstrukce budou neoptimální nebo přinejmenším podivné, avšak nemělo by to mít vliv na funkčnost. Koncept celé aplikace vypadá takto.

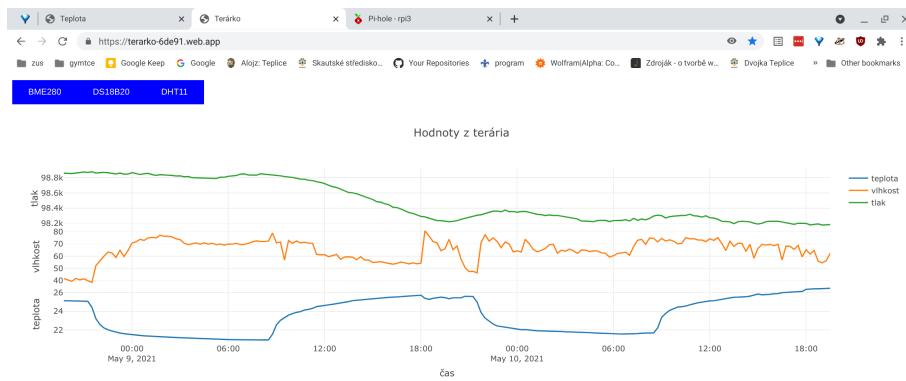


Obrázek 4.19: Průběh programu

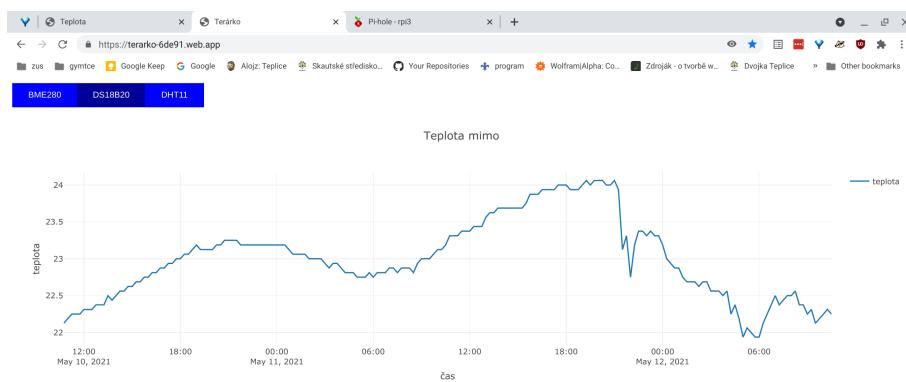
Na začátku webové stránky, která tvoří kostru celého programu, načtu Firebase knihovny. Vzhledem k tomu, že používám Firebase hosting, je tato část velmi jednoduchá, neb nemusím řešit autorizace... Poté načtu knihovnu Plotly, kterou používám pro vykreslování grafů. Následně vytvořím přepínací tlačítka a plochu pro graf. Nakonec zavolám funkci, co mi vykreslí graf hodnot v teráriu.

Do Firebase se přihlásím za použití OAuth u Googlu a to z jediného důvodu. Při nastavování Firestoru jsem nastavil pravidla tak, že bez přihlášení nikoho k datům nepustí. Takže jsem potřeboval nějak ověřit uživatele. Druhý problém byl, jak přesvědčit databázi, aby mi umožnila přístup k datům. Tady jsem použil místo řešení problémů s účty... takový špinavý trik. Po přihlášení mi Google přidělil identifikátor, kterým se prokazují vůči aplikaci, když jsem do pravidel přidal podmínu, že tento identifikátor může číst data, vše fungovalo. Není to řešení pro více uživatelů, ale dostačuje.

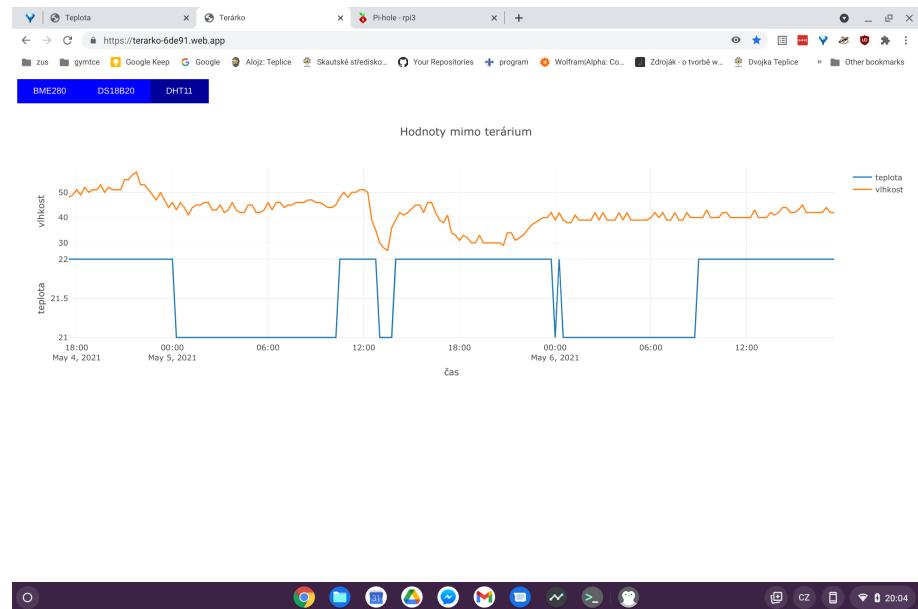
Následné vykreslení grafů už je v podstatě jen otázka, jak převést data z Firebase do pole. Začnu tím, že vyberu správnou kolekci dle použitého senzoru, tu si seřadím podle timestampu a z ní vezmu posledních 192 hodnot, což odpovídá cca posledním 48 hodinám. Následně pouze proiteruji přes všechna data a přidám je do pole hodnot, jediná výjimka je u času, který před tím převedu na čitelný tvar a do aktuálního časového pásma. Pak už data v poli pouze předám knihovně Plotly, která je vykreslít. Když mám v jednom grafu více datových polí, vykresluji je nad sebe se stejnou časovou osou, abych mohl porovnávat odpovídající hodnoty. Výsledek pak vypadá nějak takto.



Obrázek 4.20: Data z terária



Obrázek 4.21: Teplota mimo terárium



Obrázek 4.22: Teplota a vlhkost mimo terárium



Závěr

Hlavní cíl mé práce tj. udržet kytičky naživu se myslím, podařilo splnit, neb zatím žijí. Uvidím jak se jim bude dařit dál. Myslím že měření mi v lecčem pomohlo udržet je naživu, například jsem díky němu zjistil, že je málo zalévám, zvýšil jsem frekvenci a začali vypadat lépe.

Ohledně splnění cílů myslím, že vytvořené řešení splňuje až na jeden všechny hlavní požadavky, které jsme si zadal. Hodnoty se měří, ukládají se je na SD kartu a přes bránu se posílají do Firebase, odkud si je mohu zobrazit pouze já. Jediný požadavek, jež jsem nezvládl splnit, je ten na podepisování poslaných dat, na ochranu před podvržením.

Myslím, že případné zlepšování by se mohlo zaměřit na vylepšení zabezpečení cesty od senzorů na bránu, přidání statistik a možnosti zobrazení hodnot z většího intervalu či zobrazení stavu senzorů přímo u terária. Případně by se dalo zrychlit přidávání nových senzorů, pokud možno tak, že ho přidám, jednou nastavím, kde je a co měří a to bude vše.



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/@olourme/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/@olourme/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/@olourme/our-iot-journey-through-esp8266-firebase-angular-and-plotly-js-part-3-644048e90ca4>.
- PÁJENÍČKO, 2020a. *Digitální čidlo teploty Dallas DS18B20* [online] [cit. 2021-05-13]. Dostupné z: <https://pajenicko.cz/digitalni-cidlo-teploty-dallas-ds18b20>.
- PÁJENÍČKO, 2020b. *Senzor na měření teploty, vlhkosti a tlaku BME280* [online] [cit. 2021-05-13]. Dostupné z: <https://pajenicko.cz/senzor-na-mereni-teploty-vlhkosti-tlaku-bme280-ext>.
- PÁJENÍČKO, 2020c. *Senzor teploty a vlhkosti vzduchu DHT11* [online] [cit. 2021-05-13]. Dostupné z: <https://pajenicko.cz/senzor-teploty-vlhkosti-vzduchu-dht11>.
- RUEL, Marc-Antoine, 2021. *periph* [online] [cit. 2021-05-13]. Dostupné z: <https://periph.io/>.
- SIRUPSEN, 2021. *Logrus* [online] [cit. 2021-05-13]. Dostupné z: <https://github.com/sirupsen/logrus>.
- STEHLÍK, Petr, 2019. *ESP32 je tu. Co přinese nástupce ESP8266?* [online] [cit. 2021-05-13]. Dostupné z: <https://www.root.cz/clanky/esp32-je-tu-co-prinese-nastupce-esp8266/>.
- TIŠŇOVSKÝ, Pavel, 2018. *Rozhraní, metody, gorutiny a kanály v programovacím jazyku Go* [online] [cit. 2021-05-13]. Dostupné z: <https://www.root.cz/clanky/rozhrani-metody-gorutiny-a-kanaly-v-programovacim-jazyku-go/>.
- 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/>.
- WIKIPEDIA, 2021a. *1-Wire — Wikipedia, The Free Encyclopedia* [online] [cit. 2021-05-13]. Dostupné z: <https://en.wikipedia.org/w/index.php?title=1-Wire&oldid=998430808>.
- WIKIPEDIA, 2021b. *Firebase — Wikipedia, The Free Encyclopedia* [online] [cit. 2021-05-13]. Dostupné z: <https://en.wikipedia.org/w/index.php?title=Firebase&oldid=1022309186>.
- WIKIPEDIA, 2021c. *Single-page application — Wikipedia, The Free Encyclopedia* [online] [cit. 2021-05-13]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Single-page_application&oldid=1022829502.
- WIKIPEDIE, 2016. *Secure Sockets Layer — Wikipedie: Otevřená encyklopédie* [online] [cit. 2021-05-13]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Secure_Sockets_Layer&oldid=14275770.
- WIKIPEDIE, 2018. *Unixový čas — Wikipedie: Otevřená encyklopédie* [online] [cit. 2021-05-13]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Unixov%C3%A1_%C4%8D%C5%BD_%C4%8Das%C5%99&oldid=16539952.
- WIKIPEDIE, 2019a. *Garbage collection — Wikipedie: Otevřená encyklopédie* [online] [cit. 2021-05-13]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Garbage_collection&oldid=17400397.
- WIKIPEDIE, 2019b. *Gateway — Wikipedie: Otevřená encyklopédie* [online] [cit. 2021-05-13]. Dostupné z: <https://cs.wikipedia.org/w/index.php?title=Gateway&oldid=17400456>.
- WIKIPEDIE, 2020a. *CSV — Wikipedie: Otevřená encyklopédie* [online] [cit. 2021-05-13]. Dostupné z: <https://cs.wikipedia.org/w/index.php?title=CSV&oldid=18612073>.
- WIKIPEDIE, 2020b. *Digitální certifikát — Wikipedie: Otevřená encyklopédie* [online] [cit. 2021-05-13]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Digit%C3%A1ln%C3%AD_certifik%C3%A1t&oldid=18019029.
- WIKIPEDIE, 2020c. *Git — Wikipedie: Otevřená encyklopédie* [online] [cit. 2021-05-13]. Dostupné z: <https://cs.wikipedia.org/w/index.php?title=Git&oldid=18596527>.
- WIKIPEDIE, 2020d. *GitHub — Wikipedie: Otevřená encyklopédie* [online] [cit. 2021-05-13]. Dostupné z: <https://cs.wikipedia.org/w/index.php?title=GitHub&oldid=19204041>.

- WIKIPEDIE, 2020e. *JavaScript Object Notation* — Wikipedie: Otevřená encyklopédie [online] [cit. 2021-05-13]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=JavaScript_Object_Notation&oldid=19182922.
- WIKIPEDIE, 2020f. *Knihovna (programování)* — Wikipedie: Otevřená encyklopédie [online] [cit. 2021-05-13]. Dostupné z: [https://cs.wikipedia.org/w/index.php?title=Knihovna_\(programov%C3%A1n%C3%AD\)%5C&oldid=19146095](https://cs.wikipedia.org/w/index.php?title=Knihovna_(programov%C3%A1n%C3%AD)%5C&oldid=19146095).
- WIKIPEDIE, 2020g. *Linker* — Wikipedie: Otevřená encyklopédie [online] [cit. 2021-05-13]. Dostupné z: <https://cs.wikipedia.org/w/index.php?title=Linker%5C&oldid=19068646>.
- WIKIPEDIE, 2020h. *NoSQL* — Wikipedie: Otevřená encyklopédie [online] [cit. 2021-05-13]. Dostupné z: <https://cs.wikipedia.org/w/index.php?title=NoSQL%5C&oldid=19045689>.
- WIKIPEDIE, 2020i. *OAuth* — Wikipedie: Otevřená encyklopédie [online] [cit. 2021-05-14]. Dostupné z: <https://cs.wikipedia.org/w/index.php?title=OAuth%5C&oldid=18362750>.
- WIKIPEDIE, 2021a. *Go (programovací jazyk)* — Wikipedie: Otevřená encyklopédie [online] [cit. 2021-05-13]. Dostupné z: [https://cs.wikipedia.org/w/index.php?title=Go_\(programovac%C3%AD_jazyk\)%5C&oldid=19631327](https://cs.wikipedia.org/w/index.php?title=Go_(programovac%C3%AD_jazyk)%5C&oldid=19631327).
- WIKIPEDIE, 2021b. *GPIO* — Wikipedie: Otevřená encyklopédie [online] [cit. 2021-05-13]. Dostupné z: <https://cs.wikipedia.org/w/index.php?title=GPIO%5C&oldid=19453790>.
- WIKIPEDIE, 2021c. *I²C* — Wikipedie: Otevřená encyklopédie [online] [cit. 2021-05-13]. Dostupné z: <https://cs.wikipedia.org/w/index.php?title=I%C2%5C%5C&oldid=19796653>.
- WIKIPEDIE, 2021d. *Křížový překladač* — Wikipedie: Otevřená encyklopédie [online] [cit. 2021-05-13]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=K%C5%5C%5C%99%C3%5C%AD%5C%C5%5C%BEov%C3%5C%BD_p%5C%C5%5C%99eklada%C4%5C%8D%5C&oldid=19334589.
- WIKIPEDIE, 2021e. *Raspberry Pi* — Wikipedie: Otevřená encyklopédie [online] [cit. 2021-05-13]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Raspberry_Pi%5C&oldid=19806914.
- WIKIPEDIE, 2021f. *Webhosting* — Wikipedie: Otevřená encyklopédie [online] [cit. 2021-05-13]. Dostupné z: <https://cs.wikipedia.org/w/index.php?title=Webhosting%5C&oldid=19555621>.

Seznam obrázků

2.1	Takto přibližně potečou data	14
3.1	Raspberry pi 2	15
3.2	Modul s BME280, stříbrný čtvereček uprostřed	16
3.3	DS18B20, pouzdro TO-92	17
3.4	DHT11, modrý obdélníček	18
4.1	Průběh programu	21
4.2	Firebase, krok 1	24
4.3	Firebase, krok 2	24
4.4	Firebase, krok 3	25
4.5	Firebase, hotovo	25
4.6	Firebase	26
4.7	Firestore, krok 1	26
4.8	Firestore, krok 2	27
4.9	Firestore	27
4.10	Přidání aplikace	28
4.11	Jméno aplikace	29
4.12	Firestore SDK	29
4.13	Propojení s hostingem	30
4.14	Vybrání hostingu	30
4.15	Zjednodušený kód	31
4.16	Průběh programu	31
4.17	Vygenerování klíče, krok 1	32
4.18	Vygenerování klíče, krok 2	32
4.19	Průběh programu	33
4.20	Data z terária	34
4.21	Teplota mimo terárium	34
4.22	Teplota a vlhkost mimo terárium	35

Seznam tabulek

3.1	Parametry BME280	17
3.2	Parametry DS18B20	18
3.3	Parametry DHT11	19



Slovníček pojmu

brána Centrální bod, kam senzory s celé domácnosti odesílají data. Zde se rozhoduje, co s nimi a případně se posílají dál. Případné akční prvky (například žárovka) by braly informace odtud. (Wikipedie, 2019b) 13, 23, 37

certifikát Digitálně podepsaný veřejný šifrovací klíč, kterým někdo nebo něco dokazuje, že je tím za koho se vydává. (Wikipedie, 2020b) 13

cross-kompilace Překlad programu pro jinou platformu, než na které je překládán. (Wikipedie, 2021d) 21

Firebase Platforma pro vytváření mobilních a webových aplikací provozovaná Googlem. Začala v roce 2011 jako samostatná společnost, v roce 2014 ji Google koupil. Ten jí nabízí jako svůj hlavní produkt pro vývoj těchto aplikací. Firebase je nadstavbová aplikace nad Google Cloud Platform, avšak uživateli od ní úplně odstíní a umožňuje komfortně tvořit aplikace bez nutnosti detailně zkoumat jak cloud funguje. (Wikipedia, 2021b) 23, 26, 28–31, 33, 34, 37

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. (Wikipedie, 2019a) 46

Git Git je v informatice distribuovaný systém správy verzí vytvořený Linusem Torvaldsem pro vývoj jádra Linuxu. Návrh Gitu byl ovlivněn projekty BitKeeper (dříve používán pro vývoj jádra Linuxu) a Monotone. V současnosti (2016) je projekt Git používán mnoha známými projekty, například: jádro Linuxu, X.Org nebo Ruby on Rails. Projekt spravuje Junio Hamam, je šířen pod GPL verze 2 a jedná se o svobodný software. (Wikipedie, 2020c) 45

GitHub GitHub je webová služba podporující vývoj softwaru za pomocí verzovacího nástroje Git. GitHub nabízí bezplatný hosting pro open source projekty. V roce 2018 ho koupil Microsoft. (Wikipedie, 2020d) 28

Go Kompilovaný staticky typovaný jazyk od Googlu, 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. (Wikipedie, 2021a) 21, 23, 31, 47

gorutina Odlehčené vlákno (Tišňovský, 2018) 21, 23

hosting Pronájem prostoru pro webové stránky na cizím serveru. (Wikipedie, 2021f) 28–31, 45

I²C Sériová sběrnice od firmy Philips, používaná pro připojení nízko rychlostních periferií. Používá čtyři dráty: napětí, zem, hodiny a datový kabel. (Wikipedie, 2021c) 16

JSON Datový formát založený na syntaxi javascriptu, oblíbený zejména ve webových aplikacích. (Wikipedie, 2020e) 23

knihovna Soubor funkcí, tříd či konstant, které jsou využívány více programy. (Wikipedie, 2020f) 11, 21, 22, 30, 32–34

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 přibalenы k výslednému spustitelnému souboru. (Wikipedie, 2020g) 21

message broker 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 třeba naměřené teploty a on ji přepoše 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. (Tišňovský, 2019b) 19, 23

NoSQL Databázový koncept, který nepoužívá tabulky, ale ukládá data jinak. Například jako dvojice klíč-hodnota, nebo třeba jako stromovou strukturu. (Wikipedie, 2020h) 26

OAuth OAuth je otevřený protokol, navržený Blainem Cookem a Chrisem Messinou. Cílem je poskytnout bezpečnou autentizaci a autorizaci oproti API různých služeb, například Googlu, Facebooku..., a to jednotně pro desktopové, mobilní i webové aplikace. (Wikipedie, 2020i) 33

OneWire Sériová sběrnice používaná výhradně firmou Dallas pro připojení jejích produktů. Stačí jí pouze tři dráty: napětí, zem a datový s tím, že lze odstranit napájecí kabel a senzor je poté napájen paraziticky s datového kabelu. (Wikipedia, 2021a) 17, 18

periph.io Knihovna pro jazyk Go, která se snaží zapouzdřit komunikaci po GPIO a poskytuje rozhraní pro oblíbené senzory, aby jejich použití bylo co nejjednodušší. (Ruel, 2021) 22

Plotly Javascriptová knihovna pro vykreslování grafů. (Tišňovský, 2020) 33, 34

Single-page application Webová aplikace která s uživatelem reaguje pomocí dynamického překreslování jediné stránky místo načítání nových. Jejím cílem je vytvořit zdání nativní aplikace. (Wikipedia, 2021c) 33

SSL Protokol vytvářející šifrovanou cestu mezi transportní a aplikační vrstvou. Dnes se místo něj používá protokol TLS, avšak stále se používá označení SSL. (Wikipedie, 2016) 13

timestamp Počet sekund/něco s větší přesností od 1.1. 1970 (Wikipedie, 2018) 22, 23, 33, 34



Akrony whole

CSV comma separated values 22

GPIO general purpose input/output 22, 47



Příloha A

Zdrojový kód

A.1 Měřící stanice

A.1.1 global.go

```
package main

import (
    "time"

    "periph.io/x/conn/v3/physic"
    stan "github.com/nats-io/stan.go"
)

type measure struct {
    sense      physic.Env
    err        error
    timestamp time.Time
}

type BME280Msg struct {
    Timestamp int64 `json:"timestamp"`
    Temperature float64 `json:"temperature"`
    Humidity float64 `json:"humidity"`
    Pressure float64 `json:"pressure"`
}

type DS18B20Msg struct {
    Timestamp int64 `json:"timestamp"`
    Temperature float64 `json:"temperature"`
}
```

```
type DHT11Msg struct {
    Timestamp  int64   `json:"timestamp"`
    Temperature float64 `json:"temperature"`
    Humidity    float64 `json:"humidity"`
}

const (
    logFile = "/var/log/measures-terarko.log"
    //logFile = "log"
    csvFilePrefix = "/home/pi/data/terarko-"
    subjectPrefix = "terarko"
    MAXTRY        = 3
    MAX           = 120
    SENSORSCOUNT = 3
)

const (
    BME280 = iota
    DS18B20
    DHT11
)

var sensors []string = []string{"BME280", "DS18B20", "DHT11"}
```

A.1.2 measure.go

```
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 )
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 . NatsOptions{} )
    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 )
cron . NewCronJob( cron . ANY , cron . ANY , cron . ANY , cron . ANY , 15 , 10 )
cron . NewCronJob( cron . ANY , cron . ANY , cron . ANY , cron . ANY , 30 , 10 )
cron . NewCronJob( cron . ANY , cron . ANY , cron . ANY , cron . ANY , 45 , 10 )
// cron . NewCronJob( cron . ANY , cron . ANY , cron . ANY , cron . ANY , cron . ANY )
log . Debug( "Set_CRON" )
<-forever
}

```

A.1.3 getData.go

```

package main

import (
    "fmt"
    "time"

    "github . com / prokopparuzek / go-dht"
    "github . com / yryz / ds18b20"
    "periph . io / x / conn / v3 / i2c / i2creg"
)

```

```
"periph.io/x/conn/v3/physic"
"periph.io/x/devices/v3/bmxx80"

    log "github.com/sirupsen/logrus"
)

func getBME280() (e physic.Env, err error) {
    b, err := i2creg.Open("")
    if err != nil {
        log.Error(err)
        return
    }
    defer b.Close()
    dev, err := bmxx80.NewI2C(b, 0x77, &bmxx80.DefaultOpts)
    if err != nil {
        return
    }
    defer dev.Halt()
    dev.Sense(&e)
    return e, nil
}

func getDS18B20() (e physic.Env, err error) {
    sensors, err := ds18b20.Sensors()
    if err != nil {
        return
    }
    t, err := ds18b20.Temperature(sensors[0])
    if err != nil {
        return
    }
    e.Temperature.Set(fmt.Sprintf("%fC", t))
    return
}

func getDHT11() (e physic.Env, err error) {
    dht, err := dht.NewDHT("GPIO17", dht.Celsius, "dht11")
    if err != nil {
        log.Error(err)
        return
    }
    h, t, err := dht.ReadRetry(10)
    if err != nil {
        return
    }
    e.Temperature.Set(fmt.Sprintf("%fC", t))
    e.Humidity.Set(fmt.Sprintf("%f%%", h))
```

```

    return
}

func getMeasure() (out [SENSORSCOUNT] measure) {
    var e physic.Env
    var err error
    var logger *log.Entry
    var functions [SENSORSCOUNT]func() (physic.Env, error)

    functions[BME280] = getBME280
    functions[DS18B20] = getDS18B20
    functions[DHT11] = getDHT11

    for i := 0; i < SENSORSCOUNT; i++ {
        logger = log.WithField("sensor", sensors[i])
        for j := 0; j < MAXTRY; j++ {
            e, err = functions[i]()
            if err != nil {
                logger.Error(err)
                continue
            } else {
                logger.Debug("Read successfully")
                break
            }
        }
        if err != nil {
            logger.Error("Too many errors")
        }
        logger.Debug(e)
        out[i] = measure{e, err, time.Now()}
    }
    return
}

```

A.1.4 csvSave.go

```

package main

import (
    "encoding/csv"
    "os"
    "log" "github.com/sirupsen/logrus"
)

func saveData(what string, data [string]) {
    f, err := os.OpenFile(csvFilePrefix+what+".csv", os.O_APPEND|os.O_CREATE, 0644)
    defer f.Close()
    writer := csv.NewWriter(f)
    defer writer.Flush()
    for _, v := range data {
        if err := writer.Write([]string{v}); err != nil {
            log.Panicf("Error writing to file: %v", err)
        }
    }
}

```

```
    if err != nil {
        log.Panicln(err)
    }
    defer f.Close()
    log.WithField("file", what).Debug("Open_file")
    writer := csv.NewWriter(f)
    writer.Write(data)
    writer.Flush()
    err = writer.Error()
    if err != nil {
        log.Panicln(err)
    }
    log.Debug("Stored_csv")
}

func saveCsv(toCsv [SENSORSCOUNT][]string) {
    for i, data := range toCsv {
        saveData(sensors[i], data)
    }
}
```

A.1.5 sendData.go

```
package main

import (
    "encoding/json"
    "fmt"
    "time"

    "periph.io/x/conn/v3/physic"
    log "github.com/sirupsen/logrus"
)

func sendMsg(Jmsg []byte, subjectSuffix string, logger *log.Entry) {
    var tries int
    var err error

    for tries = 0; tries < MAX; tries++ {
        err = scon.Publish(subjectPrefix+":"+subjectSuffix, Jmsg)
        if err != nil {
            logger.Debug("Error, will retry")
            time.Sleep(60 * time.Second)
        } else {
            logger.Debug("Deliver")
            break
        }
    }
}
```

```

        }
    }
    if tries >= MAX {
        logger.Error(err, "Cannot deliver")
    }
}

func sendMeasures(_ time.Time) {
    var sense measure
    var toCsv [SENSORSCOUNT][]string
    data := getMeasure()

    // BME280
    var msgB BME280Msg
    sense = data[BME280]
    if sense.err == nil {
        msgB.Timestamp = sense.timestamp.Unix()
        msgB.Humidity = float64(sense.sense.Humidity) / float64(100)
        msgB.Temperature = sense.sense.Temperature.Celsius()
        msgB.Pressure = float64(sense.sense.Pressure) / float64(100)
        Jmsg, err := json.Marshal(msgB)
        logger := log.WithField("message", "BME280"+string(Jmsg))
        if err == nil {
            toCsv[BME280] = []string{fmt.Sprintf(msgB.Timestamp)}
            go sendMsg(Jmsg, "BME280", logger)
        } else {
            logger.Error(err)
        }
    }
    // DS18B20
    var msgD DS18B20Msg
    sense = data[DS18B20]
    if sense.err == nil {
        msgD.Timestamp = sense.timestamp.Unix()
        msgD.Temperature = sense.sense.Temperature.Celsius()
        Jmsg, err := json.Marshal(msgD)
        logger := log.WithField("message", "DS18B20"+string(Jmsg))
        if err == nil {
            toCsv[DS18B20] = []string{fmt.Sprintf(msgD.Timestamp)}
            go sendMsg(Jmsg, "DS18B20", logger)
        } else {
            logger.Error(err)
        }
    }
    //DHT11
    var msgT DHT11Msg
    sense = data[DHT11]
}

```

```
if sense.err == nil {
    msgT.Timestamp = sense.timestamp.Unix()
    msgT.Temperature = sense.sense.Temperature.Celsius()
    msgT.Humidity = float64(sense.sense.Humidity) / float64(ph)
    Jmsg, err := json.Marshal(msgT)
    logger := log.WithField("message", "DHT11"+string(Jmsg))
    if err == nil {
        toCsv[DHT11] = []string{fmt.Sprint(msgT.Timestamp),
            go sendMsg(Jmsg, "DHT11", logger)
    } else {
        logger.Error(err)
    }
}
saveCsv(toCsv)
}
```

A.2 Brána

A.2.1 fire.go

```
package main

import (
    "context"
    "encoding/json"
    "os"
    "time"

    "cloud.google.com/go/firestore"
    firebase "firebase.google.com/go"
    stan "github.com/nats-io/stan.go"
    log "github.com/sirupsen/logrus"
    "google.golang.org/api/option"
)

//const logFile = "/var/log/fire.log"
const logFile = "log"

var client *firestore.Client
var ctx context.Context

var subjects []string = []string{"terarko:BME280", "terarko:DHT11", "terarko:MQTT"}
```

```
func handleMsg(msg *stan.Msg) {
    var payload map[string]interface{}
    payload = make(map[string]interface{})
```

```

        ref := client.Collection(msg.Subject)
        json.Unmarshal(msg.Data, &payload)
        log.WithField("payload", payload).Debug()
        _, _, err := ref.Add(ctx, payload)
        if err != nil {
            log.Println(err)
            return
        }
        log.Debug("Fired")
        msg.Ack()
    }

func main() {
    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, 0644)
    if err != nil {
        log.WithField("file", logFile).Error(err)
    } else {
        log.SetOutput(f)
    }
    forever := make(chan bool)
    var sc stan.Conn
    for {
        sc, err = stan.Connect("measures", "terarko-gun", stan.Option{
            Context: context.Background(),
            Transport: &http.Transport{
                MaxConnsPerHost: 100,
                MaxIdleConns:    100,
                IdleConnTimeout: 10 * time.Second,
            },
            DialTLS: func(addr string) (net.Conn, error) {
                conn, err := net.Dial("tcp", addr)
                if err != nil {
                    return nil, err
                }
                return &tlsConn{conn: conn}, nil
            },
        })
        if err != nil {
            log.Error(err)
            time.Sleep(time.Second * 30)
            continue
        }
        break
    }
    defer sc.Close()
    log.Debug("Connected")
    // firebase
    ctx = context.Background()
    opt := option.WithCredentialsFile("service.json")
    app, err := firebase.NewApp(ctx, nil, opt)
    if err != nil {
        log.Fatalf("error initializing app: %v\n", err)
    }
    log.Debug("Connected_to_firebase")
    client, err = app.Firestore(ctx)
    if err != nil {
        log.Fatalln(err)
    }
}

```

```
        }
    defer client.Close()
    log.Debug("Connected_to_firestore")
    for _, s := range subjects {
        _, err = sc.Subscribe(s, handleMsg, stan.DurableName("2"),
        if err != nil {
            log.Error(err)
        }
    }
    log.Debug("Subscribed")
    <-forever
}
```

A.3 Webová aplikace

A.3.1 index.html

```
<style>
.btn-group button {
    background-color: blue; /* Green background */
    border: 1px solid blue; /* Green border */
    color: white; /* White text */
    padding: 10px 24px; /* Some padding */
    cursor: pointer; /* Pointer/hand icon */
    float: left; /* Float the buttons side by side */
}

/* Clear floats (clearfix hack) */
.btn-group:after {
    content: "";
    clear: both;
    display: table;
}

.btn-group button:not(:last-child) {
    border-right: none; /* Prevent double borders */
}

/* Add a background color on hover */
.btn-group button:hover {
    background-color: #000099;
}
</style>
<head>
    <titletitle>
<!-- The core Firebase JS SDK is always required and must be listed first -->
```

```

<script src="/__/firebase/8.4.3.firebaseio-app.js"></script>

<!-- TODO: Add SDKs for Firebase products that you want to use
      https://firebase.google.com/docs/web/setup#available-libraries -->
<script src="/__/firebase/8.4.3.firebaseio-analytics.js"></script>

<!-- Add Firebase products that you want to use -->
<!-- test -->
<script src="/__/firebase/8.4.3.firebaseio-auth.js"></script>
<script src="/__/firebase/8.4.3.firebaseio-firebase.js"></script>
<script src="/__/firebase/init.js"></script>
<!-- plotly -->
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
</head>
<body>
  <div class="btn-group">
    <button onclick="plotBME280()">BME280</button>
    <button onclick="plotDS18B20()">DS18B20</button>
    <button onclick="plotDHT11()">DHT11</button>
  </div>
  <div id="graph"></div>
  <script src="./plot.js"></script>
  <!-- default chart -->
  <script>plotBME280()</script>
</body>

```

A.3.2 plot.js

```

// auth
var hasLocalStorageUser;
firebase.auth().getRedirectResult().then(function() {
  sessionStorage.setItem("User", "OK");
  hasLocalStorageUser = "OK";
});
firebase.auth().setPersistence(firebase.auth.Auth.Persistence.SESSION);
if (!firebase.auth().currentUser) {
  hasLocalStorageUser = sessionStorage.getItem("User");
  if (hasLocalStorageUser != "OK") {
    var provider = new firebase.auth.GoogleAuthProvider();
    provider.addScope('https://www.googleapis.com/auth/plus.login');
    firebase.auth().signInWithRedirect(provider);
  }
}

if (hasLocalStorageUser == "OK") {
  // Firestore
  var db = firebase.firestore();
}

```

```
    firebase.firestore().enablePersistence();
}

function plotBME280() {
  var cur = db.collection('terarko:BME280');
  cur = cur.orderBy('timestamp').limitToLast(192);
  var x = [];
  var temp = [];
  var hum = [];
  var press = [];
  var layout;
  var config;
  cur.get().then((docs) => {
    docs.forEach((doc) => {
      var date = new Date(doc.data().timestamp * 1000)
      date.setMinutes(date.getMinutes() - date.getTimezoneOffset())
      x.push(date.toISOString())
      temp.push(doc.data().temperature);
      hum.push(doc.data().humidity);
      press.push(doc.data().pressure);
    });
    temp = [
    {
      type: "Scattergl",
      mode: "lines",
      name: 'teplota',
      x: x,
      y: temp
    },
    {
      type: "Scattergl",
      mode: "lines",
      name: 'vlhkost',
      yaxis: 'y2',
      x: x,
      y: hum
    },
    {
      type: "Scattergl",
      mode: "lines",
      name: 'tlak',
      yaxis: 'y3',
      x: x,
      y: press
    }
  ];
  layout = {
```

```

        title: 'Hodnoty z terária',
        showlegend: true,
        xaxis: {
            title: 'čas',
        },
        yaxis: {
            title: 'teplota',
            domain: [0, 0.33]
        },
        yaxis2: {
            title: 'vlhkost',
            domain: [0.33, 0.66]
        },
        yaxis3: {
            title: 'tlak',
            domain: [0.66, 1]
        },
    };
    config = {
    };
    Plotly.newPlot('graph', temp, layout, config);
});
}

function plotDS18B20() {
    var cur = db.collection('terarko:DS18B20');
    cur = cur.orderBy('timestamp').limitToLast(192);
    var x = [];
    var temp = [];
    var layout;
    var config;
    cur.get().then((docs) => {
        docs.forEach((doc) => {
            var date = new Date(doc.data().timestamp * 1000)
            date.setMinutes(date.getMinutes() - date.getTimezoneOffset())
            x.push(date.toISOString())
            temp.push(doc.data().temperature);
        });
        temp = [
        {
            type: "Scattergl",
            mode: "lines",
            name: 'teplota',
            x: x,
            y: temp
        },
        ];
    });
}

```

```
layout = {
    title: 'Teplota_mimo',
    showlegend: true,
    xaxis: {
        title: 'čas',
    },
    yaxis: {
        title: 'teplota',
    },
};

config = {
};

Plotly.newPlot('graph', temp, layout, config);
});

}

function plotDHT11() {
var cur = db.collection('terarko:DHT11');
cur = cur.orderBy('timestamp').limitToLast(192);
var x = [];
var temp = [];
var hum = [];
var layout;
var config;
cur.get().then((docs) => {
    docs.forEach((doc) => {
        var date = new Date(doc.data().timestamp * 1000)
        date.setMinutes(date.getMinutes() - date.getTimezoneOffset())
        x.push(date.toISOString())
        temp.push(doc.data().temperature);
        hum.push(doc.data().humidity);
    });
    temp = [
    {
        type: "Scattergl",
        mode: "lines",
        name: 'teplota',
        x: x,
        y: temp
    },
    {
        type: "Scattergl",
        mode: "lines",
        name: 'vlhkost',
        yaxis: 'y2',
        x: x,
        y: hum
    }
];
});
```

```
        } ,
    ] ;
layout = {
    title: 'Hodnoty mimo terárium',
    showlegend: true ,
    xaxis: {
        title: 'čas' ,
    } ,
    yaxis: {
        title: 'teplota' ,
        domain: [0 , 0.5]
    } ,
    yaxis2: {
        title: 'vlhkost' ,
        domain: [0.5 , 1]
    } ,
} ;
config = {
};

Plotly.newPlot('graph' , temp , layout , config);
}) ;
}
```