ROČNÍKOVÁ PRÁCE

Obor: Matematicko-fyzikální seminář

Vývoj detektoru potenciálně škodlivých plynů a komunikace výsledků přes WiFi na Internet

Autor: **Martin Znamenáček**
Konzultant: **Mgr. Čestmír Mniazga**
Škola: **Gymnázium ALTIS s.r.o., Dopplerova 351, 110 00 Praha**
Kraj: **Praha**
Ročník: **8. A (oktáva), 2021/2022**

Prohlášení

Prohlašuji, že jsem svou práci vypracoval samostatně a použil jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

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) ve znění pozdějších předpisů.

V Praze dne 7. 3. 2022

Martin Znamenáček

Poděkování

Rád bych poděkoval svému konzultantovi panu Mgr. Čestmíru Mniazgovi za pomoc při vedení ročníkové práce, za cenné rady a věcné připomínky, ale i vstřícnost a myšlenku, která dala za vznik tématu celé této práce.

Anotace

Ve své práci jsem se zabýval vývojem zařízení schopného detekovat koncentraci potenciálně škodlivých plynů, jakými mohou být methan, oxid uhelnatý, či třeba ethanol.

Pracoval jsem s Arduino čipem se zabudovaným WiFi rozhraním, což mi umožnilo měřené hodnoty okamžitě sdílet přes Internet do databáze.

Taktéž jsem si vytvořil webové stránky, na kterých se v reálném čase, tak jak měřič měří, zobrazují koncentrace detekovaných plynů. Této stránce jsem potom pomocí skriptů umožnil hodnoty z databáze kopírovat a neprodleně aktualizovat. Webové stránky jsem se rovněž snažil vytvořit tak, aby byly přehledné, a pokud možno funkční.

Výsledkem pak byl relativně přesný měřič koncentrací škodlivých plynů, jenž po zapojení ke zdroji (zásuvce, baterii, či USB) a připojení k jakékoliv internetové síti, je schopen i tato měření přes Internet sdílet a umožnit tak uživateli monitorovat naměřené hodnoty odkudkoliv na zemi.

Klíčová slova

Arduino; programování; senzory; detekce škodlivých plynů; WiFi; World Wide Web; HTML; JavaScript; CSS; C; C++.

OBSAH

ÚVOD

V práci se hlavně soustředím na zdrojový kód, a to jak v podobě HTML prostředí, které umožňuje zobrazení výsledků online, tak i Arduino programu, jenž inicializuje všechny součástky a zajišťuje odesílání měření do databáze.

Zároveň rozebírám hostování webových stránek a založení databáze, k čemuž používám Google Firebase.

V závěru přicházím s možnými vylepšeními a poukazuji na nedostatky či nově vytvořené příležitosti, na kterých by se v budoucnu ještě dalo pracovat.

Práce by kromě vysvětlení fungování a tvořícího procesu zařízení měla sloužit i jako návod k sestavení a zprovoznění užitých součástek a celého online systému.

CÍLE PRÁCE

- Zapojit a zprovoznit senzory na měření plynů
- Nakalibrovat senzory, aby fungovaly co nejpřesněji
- Vytvořit online databázi pro uchování a předávání výsledků měření
- Zajistit přenos z Arduino desky na online databázi
- Vytvořit webové stránky pro zobrazení možných výsledků
- Zajistit komunikaci webové stránky s databází, a tedy i Arduino deskou samotnou
- Zajistit hostování webové stránky

PROSTŘEDÍ

Zdrojový kód pro chování měřiče byl vyvíjen na Arduino desce, zapomocí softwaru Arduino IDE.

Webové stránky byly vyvíjeny přes HTML (pro kostru), CSS (pro styl) a JavaScript (pro funkce i algoritmy), a to za pomoci editoru Sublime Text 3.

Hostování stránek bylo možné skrze nahrávání přes Node.js příkazový řádek, rozšířený o správce balíčků Npm a Chocolatey, které tak zajišťují komunikaci s Firebase - App Hosting.

SOUČÁSTKY

Celkové náklady činily něco kolem 2000 Kč.

- 1 mini počítač Arduino UNO WiFi Rev2 (1300 Kč)
- 6 různých MQ senzorů (300 Kč)
- síťový napájecí adaptér stabilizovaný na 3-12 V (350 Kč)
- jumper drátky Dupont M-F, krabička a nevodivé šroubky (100 Kč)

Arduino UNO WiFi Rev2

Jedná se o Arduino desku, jež je postavená na zbrusu novém mikrokontroléru ATmega4809. Tato nová architektura přináší řadu vylepšení, jednou z nich je WiFi modul ESP32 u-blox NINA-W13, který umožňuje připojení k bezdrátové síti, a tak i komunikaci s jakoukoliv internetovou stránkou. WiFi modul má zároveň zabudovanou anténu a komunikuje bezdrátově až na vzdálenost 400 m. Rovněž má zabudovaný kryptovací akcelerátor ECC608, jenž zabezpečuje bezdrátovou komunikaci.

Ačkoliv architektura ATmega4809 mnohdy komplikuje programování čipu, a to z důvodu nekompatibility s knihovnami ze starších architektur (většina na tento modernější čip není nastavena, a proto občas zcela nefunguje), zabudovaná WiFi jednotka umožňuje velice jednoduché zapojení a nevyžaduje jakýkoliv přídavný shield, či modul. Další výhodou je absence rušení - poněvadž je WiFi modul součástí samotné desky, nestává se, že by signál vypadával, jako se tomu často děje u přídavných WiFi jednotek, jako je ESP8266.

Arduino UNO WiFi Rev2 jsem si vybral právě pro jeho WiFi periferie, pro zjednodušení hardwarové stránky, která mi umožnila více se soustředit na software, ve kterém si připadám jistější. Zároveň jsem se tak zbavil zbytečné kabeláže (minimálně 8 pinů) a umožnil čipu nerušeně pracovat pouze s MQ měřiči.

Sada MQ senzorů

Zakoupil jsem si sadu devíti MQ senzorů, ze kterých jsem vybral celkem šest odlišných měřičů. Vybíral jsem si takové, které mi připadali pokud možno nejzajímavější, či nejpraktičtější pro domácí měření.

Tuto šestici měřičů tvořily MQ3, MQ4, MQ5, MQ7, MQ8 a MQ135, přičemž každý z nich je citlivostí (nastavenou skrze vnitřní rezistory) stavěný na to, aby nejpřesněji měřil pouze omezené množství plynů.

MQ3 - stavěný hlavně na měření ethanolu (C_2H_5OH)

MQ4 - vytvořený pro detekci methanu (CH_4) a ethanu (C_2H_6)

MQ5 - nejlépe měřící propan (C_3H_8) a butan (C_4H_{10})

MQ7 - určený k zjišťování koncentrace oxidu uhelnatého (CO)

MQ8 - specializující se na vodík (H_2)

MQ135 - designovaný k detekci acetonu (C_3H_6O) či toluenu (C_7H_8)

Ačkoliv se tyto měřiče pyšní jasným vymezením plynů, které jsou schopny v ideálních podmínkách měřit (teplota 20 °C, vlhkost 65 %, koncentrace kyslíku 21 %), všechny využívají stejného principu měření, a to včetně využitých materiálů. Každý z těchto senzorů používá k detekci tenkou vrstvu oxidu číničitého (SnO_2), k zahřátí topnou cívku z nichromu (NiCr) a pro potenciometrii pozlacenou (Au) a platinovou (Pt) měrnou elektrodu.

Z tohoto důvodu je zapotřebí brát v potaz fakt, že například zvýšená koncentrace CO se sice nejvíce projeví na měřiči MQ7, avšak i měřič MQ8, určený spíše na H_2, bude do jisté míry CO taktéž detekovat, ačkoliv je primárně kalibrovaný právě na H_2.

Je tedy třeba podotknout, že se jedná o relativně levné měřiče, které jsou určené spíše k orientačnímu měření - na únik či prudce zvýšenou koncentraci škodlivých plynů jsou ale více než dostačující, a proto se dají relativně spolehlivě používat i v domácnosti namísto zakoupených detektorů například oxidu uhelnatého.

Zároveň je třeba zajistit, že měřič nepřijde do kontaktu s vodou, s vodní párou či korodujícím plyny jako chlór (Cl_2), chlorovodík (HCl) či oxidy síry (SO_x), neb by tak mohlo dojít k poničení cínové vrstvy, a tedy k atenuaci sensitivity.

V neposlední řadě je ještě třeba zmínit, že jsou měřiče schopny velmi rychle detekovat i mírně zvýšené koncentrace měřených plynů, avšak velmi dlouhou dobu trvá, než se zpátky stabilizují, takže i několik minut po vystavení měřiče detekovatelnému plynu jsou naměřené hodnoty koncentrace stále relativně vysoké.

Zapojení MQ senzorů

Samotné zapojení je velice jednoduché, MQ senzory mají pouze 4 piny:

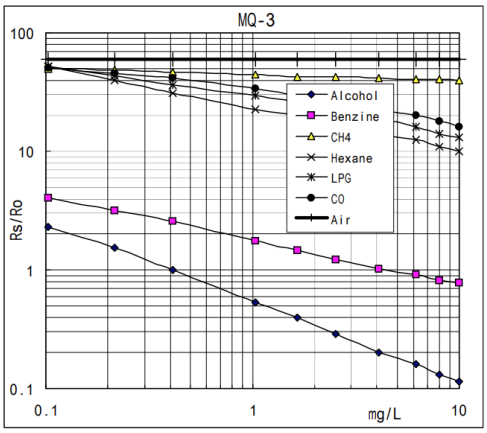
- VCC - zapojený do 5V (+)
- GND - zapojený do země (-)
- D0 - zapojený do jakéhokoliv digitálního vstupu (0-13)
- A0 - zapojený do jakéhokoliv analogového vstupu (A0-A5)

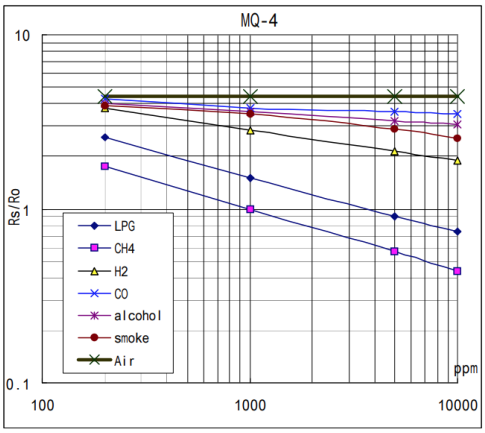
Všech 6 senzorů má identické zapojení, a tak jsem pouze užil nepájivého pole, do kterého jsem z výstupů desky vyvedl napájení (5V) a uzemnění (GND). Digitální vstupy jsem pak vyvedl rovnou na Arduino desku, do libovolných ze 14 vstupů, analogové vstupy jsem pak osadil všechny, poněvadž se jich na desce nachází právě 6.

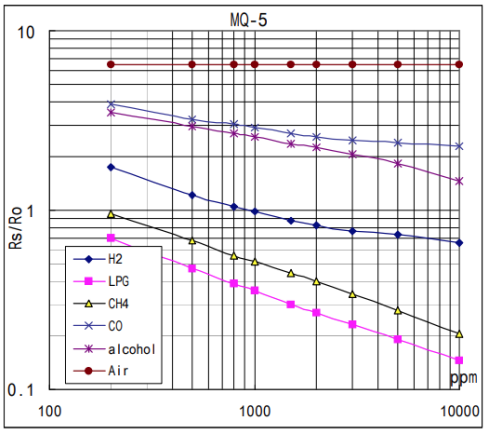
Kalibrace MQ senzorů

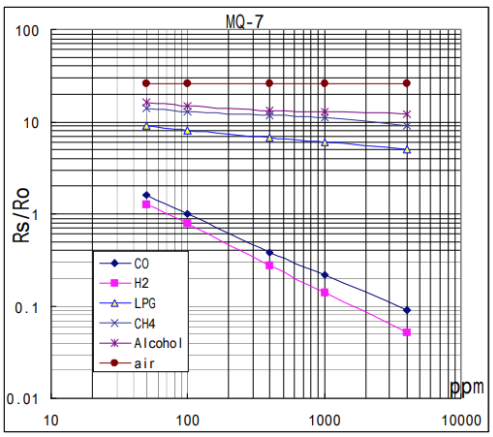
Aby MQ senzory měřily pokud možno co nejpřesněji, je potřeb je kalibrovat. Proces kalibrace lze učinit na hardwaru samotném, a to za pomoci pootáčení trimerem na měřiči, jehož pohybováním se mění elektrický odpor vnitřního měřiče.

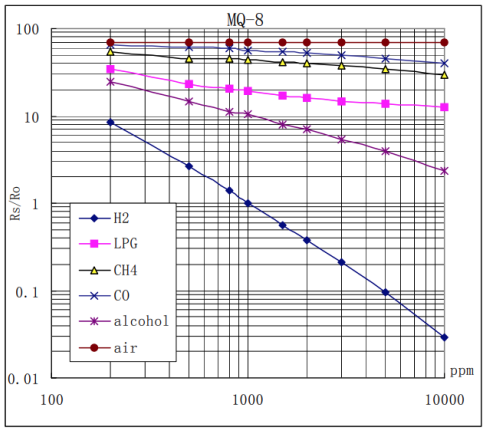
Já jsem ale zvolil metodu softwarové kalibrace. K tomu jsem potřeboval grafy senzitivity jednotlivých rezistorů. Ty jsem získal z volně dostupných online dokumentací.

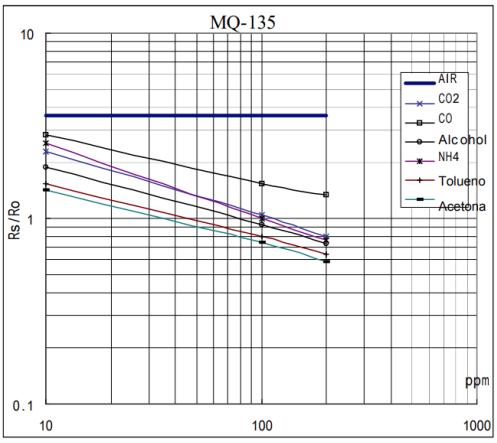












Z těchto grafů jsem pak jednoduše vyčetl maximální a minimální odpor v rozsahu koncentrace alokované pro daný měřič.

Maximální odpor nastane při stavu, kdy se v měřeném prostředí nenachází měřený plyn.

Minimální odpor naopak nastane při maximálním měřitelném nasycení plynu v měřeném prostředí.

Rozsah koncentrací jsem získal z návodu, rovněž je patrný z grafu, kdy naznačené měření začíná v nejnižší měřitelné koncentraci a končí v maximálně měřitelné koncentraci. Pouze u měřiče MQ7 jsem se od grafu mírně odchýlil a použil doporučený rozsah koncentrace, jak byl uvedený v návodu (tedy 20-2000 ppm, namísto 40-4000), a tak jsem si maximální odpor domyslel z trendu přímky.

Takto nalezené hodnoty jsem pak zanesl do tabulky:

MQ	max. R	min. R	min. konc.	max. konc.	jednotka konc.
3	2,30	0,12	100	10 000	mg/m^3
4	1,75	0,45	200	10 000	ppm
5	0,70	0,15	200	10 000	ppm
7	2,00	0,15	20	2 000	ppm
8	8,50	0,03	100	10 000	ppm
135	1,50	0,60	10	200	ppm

Tyto hodnoty mi pak umožnily spočítat můj vlastní odpor, konstantu mého měřiče, a to za pomoci jednoduchého Arduino programu.

Nejprve si nadefinujeme analogový pin, který budeme pro měřič používat

```
#define PIN_ANALOG_MQ A0
```

Pak si vytvoříme plovoucí řádovou čárku pro maximální odpor (vyčtený z tabulky) a pro počet výsledků, který v každé várce později zprůměrujeme.

```
const float maximalni_odpor = 1.5;
const byte pocet_vysledku = 250;
```

Nastavíme komunikaci se sériovým portem na základních 9600 baud.

```
void setup() {
  Serial.begin(9600);}
```

V opakující se *loop* funkci pak vytvoříme cyklus *for*, který 250krát načte hodnotu z měřiče a přičte ji do plovoucího *senzor_mereni*.

```
void loop() {
  float senzor_mereni = 0;
    for (int i = 0 ; i < pocet_vysledku ; i++) {
      senzor_mereni += analogRead(PIN_ANALOG_MQ);
      delay(5);}
```

Tuto sumu 250 načtených hodnot zprůměrujeme a převedeme na napětí senzoru, přičemž pracujeme na 5V a v 10bitovém rozlišení funkce *analogRead*, která tak umožňuje 1024 potenciálních hodnot.

```
  float senzor_napeti = senzor_mereni / pocet_vysledku * 5.0 / 1024;
```

Dále spočítáme odpor vzduchu, a to odečtením napětí senzoru od pracovního napětí (5V) a následným vydělením napětím senzoru.

```
  float odpor_vzduchu = (5.0 - senzor_napeti) / senzor_napeti;
```

Finálně získáme hledanou konstantu, kalibrovaný odpor, který je podílem odporu vzduchu a tabulkového maximálního odporu.

```
  float odpor_kalibrovany = odpor_vzduchu / maximalni_odpor;
```

Tuto konstantu potom necháme vypsat do sériového portu.

```
  Serial.print("KONSTANTA odpor_kalibrovany: ");
  Serial.println(odpor_kalibrovany);
  delay(1250);}
```

Tento proces musíme zopakovat pro všech šest měřičů, vždy pochopitelné pro jejich příslušné maximální odpory.

Je potřeba též podotknout, že měřič je schopen přesného měření až po 60-90 sekundách zahřívání, avšak při prvotní kalibraci je třeba program nechat alespoň 30 minut běžet, abychom získali co nejpřesnější a hlavně ustálenou hodnotu hledaného odporu.

Jakmile nám bude konstanta fluktuovat okolo stejného čísla (± 0.05), můžeme měření ukončit, protože jsme konstantu našli:

```
KONSTANTA odpor_kalibrovany: 1.91
KONSTANTA odpor_kalibrovany: 1.90
KONSTANTA odpor_kalibrovany: 1.90
KONSTANTA odpor_kalibrovany: 1.91
KONSTANTA odpor_kalibrovany: 1.91
KONSTANTA odpor_kalibrovany: 1.91
KONSTANTA odpor_kalibrovany: 1.92
KONSTANTA odpor_kalibrovany: 1.91
```

```
KONSTANTA odpor_kalibrovany: 1.91
KONSTANTA odpor_kalibrovany: 1.89
KONSTANTA odpor_kalibrovany: 1.90
KONSTANTA odpor_kalibrovany: 1.90
KONSTANTA odpor_kalibrovany: 1.90
KONSTANTA odpor_kalibrovany: 1.91
KONSTANTA odpor_kalibrovany: 1.91
KONSTANTA odpor_kalibrovany: 1.90
```

Ještě je třeba zmínit, že je zapotřebí kalibrovat senzory pouze za předpokladu, že všech šest těchto senzorů je zapojených do desky zároveň, neboť kalibrace pro osamocené senzory se liší, poněvadž se v mém zapojení mění napětí, které je na daný senzor alokované. Pokud je tedy zapojený pouze 1 senzor, napětí je vyšší, zatímco při všech 6 zapojených senzorech je napětí na každý ze senzorů o něco nižší, a tak se i liší hledaná konstanta.

Takto nalezené odpory, tedy kalibrační konstanty, jsem zanesl do tabulky a později využil pro výpočet koncentrace měřeného plynu.

MQ	3	4	5	7	8	135
konst. R	**1,9**	**2,9**	**12,7**	**1,5**	**0,33**	**2,1**

Vyhodnocení MQ senzorů

Nyní jsem byl připraven výsledky analogových výstupů měřičů převádět na koncentrace měřených plynů.

Nejprve jsem si zavedl jednorozměrné pole (vždy o 6 prvcích - neb je právě 6 měřičů) pro všechny potřebné proměnné, a to z hodnot vyjmutých z dvou již přiložených tabulek.

Nadefinoval jsem si tři odpory: kalibrovaný = konstanta získaná v minulém kroku - pro kalibrované, tedy přesné výsledky,

```
const float odpor_kalibrovany[6] = {1.9, 2.9, 12.7, 1.5, 0.33, 2.1};
```

maximální = odpor, ke kterému dochází při minimální/nulové koncentraci měřeného plynu,

```
const float odpor_maximalni[6] = {2.30, 1.75, 0.70, 2.00, 8.50, 1.50};
```

minimální = odpor, ke kterému dochází při maximální koncentraci měřeného plynu.

```
const float odpor_minimalni[6] = {0.12, 0.45, 0.15, 0.15, 0.03, 0.60};
```

Dále jsem si stanovil rozsah možných výsledků koncentrací - na minimum a maximum:

```
const int koncentrace_minimalni[6] = {100, 200, 200, 20, 100, 10};
const int koncentrace_maximalni[6] = {10000, 10000, 10000, 2000, 10000,
200};
```

Poté jsem si přiřadil analogové piny, ze kterých budu číst hodnoty měření:

```
const int MQ_analog_pin[6] = {A0, A1, A2, A3, A4, A5};
```

V neposlední řadě jsem si vytvořil globální pole pro jednotlivé koncentrace, abych je mohl odkudkoliv v programu adresovat (později posílat přes WiFi do databáze):

```
int koncentrace[6];
```

Pak už stačilo pouze vytvořit funkci na vyhodnocení výsledků *void MERENI_MQ(byte meric*), která bude podle stanoveného argumentu *meric* nastavovat pro daný měřič všechny potřebné výpočetní proměnné a počítat koncentrace daného měřiče.

Je nutné přečtené hodnoty z analogového pinu převést na plovoucí řádovou čárku, poněvadž nabývají relativně malých hodnot a při ponechání ve stavu *int* dochází k zaokrouhlení na 0.

Funkce *MERENI_MQ* tedy vypočítá (podobně jako u předešlé kalibrace) napětí na senzoru, to převede na odpor, a ten finálně vydělí kalibrovaným odporem, abychom získali co nejpřesnější měření.

```
void MERENI_MQ(byte meric){
  float MQ_napeti = (float)(analogRead(MQ_analog_pin[meric])) / 1024 * 5.0;
  float odpor_MQ = (5.0 - MQ_napeti) / MQ_napeti;
  float odpor_MQ_upraveny = odpor_MQ / odpor_kalibrovany[meric] * 100;
```

Taktéž je třeba podotknout, že je dobré tento odpor zvětšit, například vynásobit 100, aby se přesněji dal mapovat (podrobněji vysvětleno později).

Vytyčíme *if* podmínku, ve které porovnáme upravený odpor s odporem maximálním, při kterém je nejnižší koncentrace plynu, abychom zajistili, že pracujeme pouze v rozsahu, který měřič dovoluje.

Else podmínka pak nastaví jakoukoliv potenciálně nižší hodnotu nežli měřitelné minimum právě na minimum, abychom zamezili v případě neideálního prostředí záporným hodnotám.

```
  if ( odpor_MQ_upraveny <= ( odpor_maximalni[meric] * 100 ) ){
    koncentrace[meric] = map(odpor_MQ_upraveny, odpor_maximalni[meric] *
    100, odpor_minimalni[meric] * 100, koncentrace_minimalni[meric],
    koncentrace_maximalni[meric]);}
  else{
    koncentrace[meric] = koncentrace_minimalni[meric];}}
```

Důležité je vysvětlit funkci *map()*. Tato funkce pracuje s 5 argumenty. Nejprve jí definujeme hodnotu, kterou bude převádět, v našem případě *odpor*. Dále nastavíme potenciální *minimum* a *maximum*, kterých může *odpor* nabývat. Protože tato funkce pracuje nepřesně s malými desetinnými čísly, vynásobil jsem jak *odpor*, tak i jeho *minimum* a *maximum* číslem 100, abych se této nepřesnosti pokud možno zbavil. Tato funkce potřebuje ještě další 2 argumenty, na které bude doslova mapovat v našem případě hodnotu *odpor*. Těmito hodnotami jsou *minimum* a *maximum koncentrace*, na které je měřič stavěný.

Jinými slovy, z našeho *odporu*, a z vyčtených *minim* a *maxim odporů*, tedy rozsahu *odporu*, vybere proporčně funkce *map()* hodnotu z *maxima* a *minima koncentrace*.

Pro lepší pochopení lze uvést příklad, který by mohl nastat u měřiče MQ7.
Po nastavení konstant by mapovací funkce vypadala následovně:

```
map(odpor_MQ_upraveny, 200, 15, 20, 2000);
```

Pakliže bychom za *odpor* naměřili:

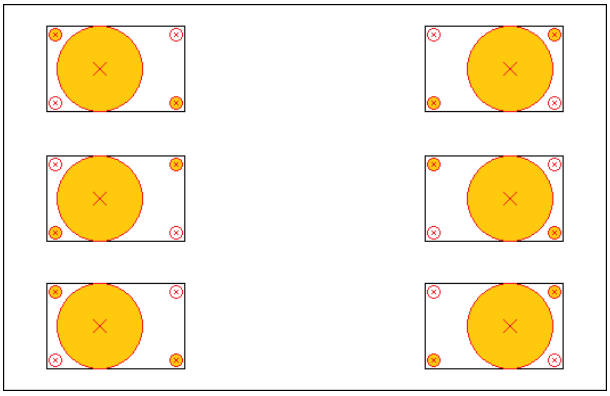
200 (to je *maximální odpor*) … výsledkem by bylo 20 (*minimální koncentrace*)
100 … výsledkem by bylo 1090
50 … výsledkem by bylo 1625
15 (to je *minimální odpor*) … výsledkem by bylo 2000 (*maximální koncentrace*)

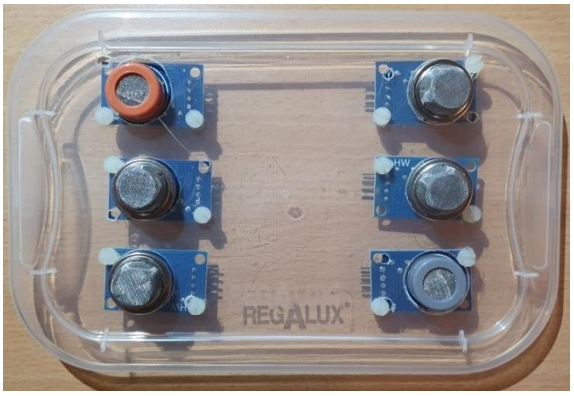
Tímto způsobem jsem byl jednoduše schopen přepočítat *odpor* na kýženou *koncentraci* a s tou poté dále pracovat.

PERIFERIE

Dále bylo potřeba vyřešit otázku, kam programovací desku a senzory umístit.

Koupil jsem si malou plastovou krabičku, do které mi tatínek vyvrtal zdířky (⌀22 mm) na umístění šesti měřičů. Pro jejich připevnění bylo zapotřebí pro každý senzor vždy alespoň po dvou malých zdířkách (⌀3 mm) pro nevodivé plastové šroubky.

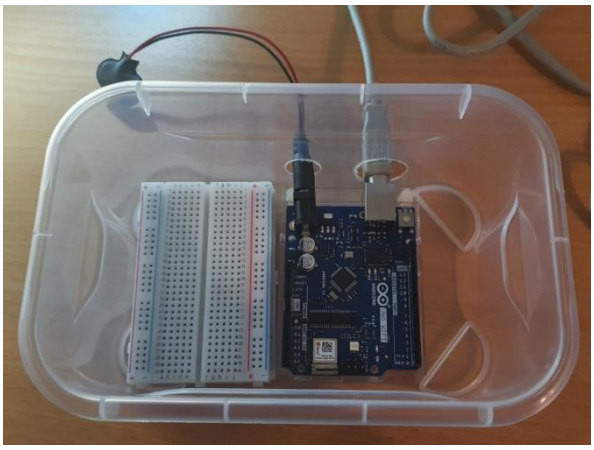


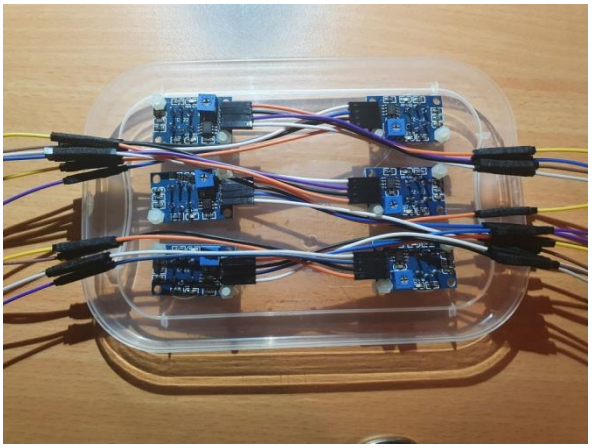


Do dna krabičky jsem pak stejnými šroubky umístil arduino desku a nepájivé pole. Do boku krabičky jsme pak ještě vyvrtali dva otvory na USB A vstup, respektive 9V napájecí adaptér.

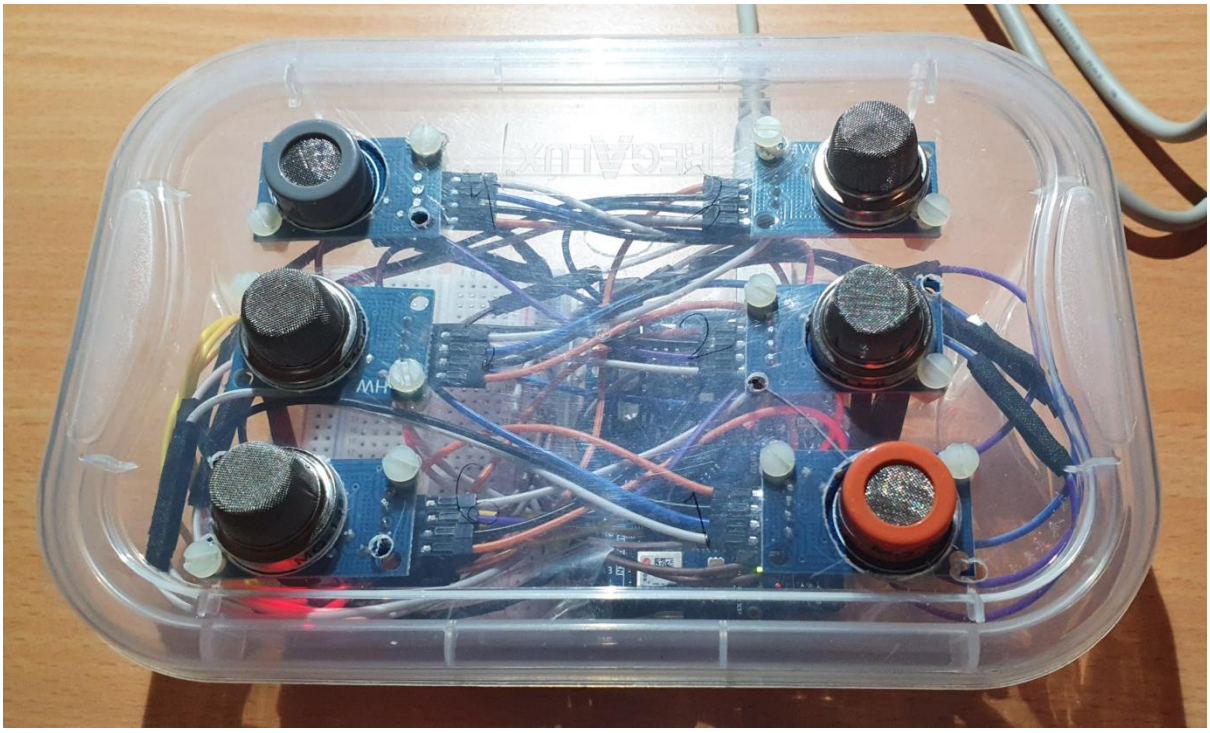
Senzory jsem rovněž osadil kabely Dupont (M-F), přičemž jsem použil klasických barev pro odlišení signálu (oranžová, zelená), uzemnění (černá/hnědá), digitálního pinu (modrá) a analogového pinu (bílá).

Do 9V napájecího pinu jsem pro testování nainstaloval adaptér na baterii, pro dlouhodobé používání je však lepší napájet desku regulovatelným síťovým napájecím adaptérem, který jsem si rovněž pořídil. Výhodou totiž je, že je stabilizovaný a disponuje přepěťovou a nadproudovou ochranu.





Po zapojení všech 24 kabelů konečný měřič vypadal následovně:



FIREBASE

Nyní, s funkčním měřicím systémem, jsem se mohl přesunout k vytvoření online databáze.

Rozhodl jsem se využít platformy Firebase od společnosti Google, poněvadž jsou jejich služby de facto zdarma (upřesnění dále) a zároveň kromě databáze rovněž poskytují hostování webových stránek, které jsem potřeboval pro finální krok.

Databáze

Vytvoření tzv. Realtime Databáze je velmi jednoduché a zdarma umožňuje uchovávat až 1 GB dat, převádět 10 GB dat měsíčně, učinit 600 tis. přepisů a 1 500 tis. přečtení měsíčně, což bylo pro tento projekt více než dostačující.

Je ale potřeba si dát pozor na jednu věc, a to na zvolení geografického umístění této databáze. Ačkoliv by z hlediska latence dávalo smysl zvolit servery například v Belgii, je potřeba použít servery ve Spojených Státech Amerických (us-central1), protože Arduino knihovny mají jinak problém s databází komunikovat, protože generují odlišné odkazy, na které Arduino knihovny nejsou stavěné.

Můj Americký odkaz byl stanoven: https://martinznamenacekdb-default-rtdb.firebaseio.com/.

Abychom mohli z databáze číst data (.read) a rovněž je zapisovat (.write), musíme nejprve upravit pravidla této databáze v obou případech na pravu (true):

```
{ "rules": { ".read": true, ".write": true }}
```

Pro zapisování je ale ještě třeba získat heslo (Database secret). V mém případě to bylo: Tug08AoxblPlTlkdjpTWMK0ZqwQfU8tm1YZxFAVt.

S tímto klíčem konečně může Arduino databázi navštívit, „odemknout“ a data přepsat, smazat, či přidat.

Připojení k síti

Nežli se ale připojíme k internetové databázi, musíme se nejprve připojit k Internetu.

Předdefinujeme si tedy identifikátor (*SSID*) a přístupové *HESLO* bezdrátové sítě WiFi, která by ideálně měla běžet na 2.4 GHz.

```
#define WIFI_SSID "wifina"
#define WIFI_HESLO "Heslo1234"
```

V nastavovacím bloku *void setup()* přiřadíme status/stav WiFi jako *IDLE* (tedy nepřipojený) a započneme cyklus while, který poběží, dokud k WiFi síti nebude vytvořeno připojení. Proces připojení se zahájí funkcí *WiFi.begin(),* do které naklademe již nadefinované *SSID* a *HESLO* sítě, ke které se snažíme připojit.

```
void setup(){
  int status = WL_IDLE_STATUS;
  while (status != WL_CONNECTED) {
    status = WiFi.begin(WIFI_SSID, WIFI_HESLO);
    delay(500);}
```

Tímto máme připojení k Internetu hotovo a můžeme přejít na posílání výsledků měřených koncentrací naší databázi.

Předání výsledků databázi

Ze všeho nejdříve musíme nastavit knihovnu pro Firebase komunikaci s modulem WiFiNINA.

```
#include <Firebase_Arduino_WiFiNINA.h>
```

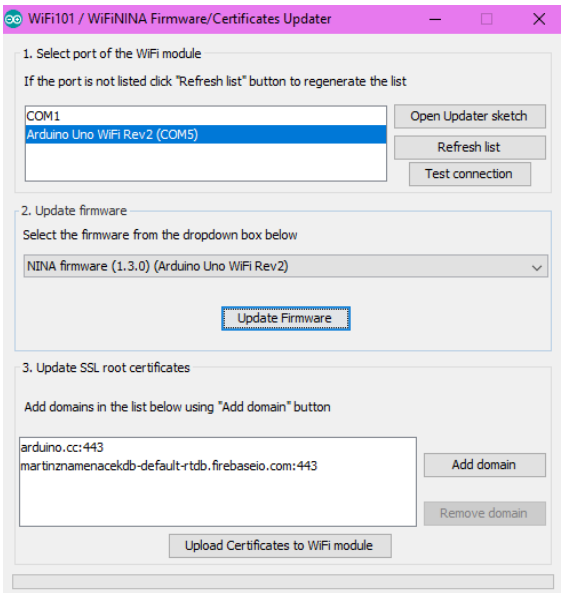
Dále je potřeba nadefinovat stránku, tedy odkaz na databázi (HOST) a posléze autorizační klíč k této databázi (AUTH):

```
#define FIREBASE_HOST "martinznamenacekdb-default-rtdb.firebaseio.com"
#define FIREBASE_AUTH "Tug08AoxblPlTlkdjpTWMK0ZqwQfU8tm1YZxFAVt"
```

Posléze musíme aktualizovat firmware WiFiNINA. To umožňuje program v nástrojích Arduino IDE > WiFi101 - WiFiNINA Firmware Updater.

Po zvolení správného portu pouze klikneme na tlačítko Update Firmware a modul WiFiNINA se automaticky aktualizuje.

Dále je nutné zadat do SSL root certifikátů odkaz (HOST) na naší Firebase databázi:



Pokud bychom tak neučinili, Firebase knihovna na Arduinu se nedokáže připojit, vždy jí bude zamítnut přístup.

Nyní databázi pojmenujeme:

```
FirebaseData databaze;
```

Pokračujeme v bloku *void setup(),* v rámci kterého započneme (.begin) připojení k Firebase databázi a nastavíme, aby se modul od databáze neodpojoval, tedy aby neustále udržoval připojení (.reconnectWiFi):

```
  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH, WIFI_SSID, WIFI_HESLO);
  Firebase.reconnectWiFi(true);
```

Konečně nastavíme, aby se vždy změřila koncentrace z každého z měřičů, a to za pomoci cyklu for, a aby se pak tato hodnota poslala do databáze, k čemuž slouží funkce *setFloat()*. Do této funkce zavedeme naši databázi a vždy k odkazu přidáme název proměnné, pod kterým chceme koncentraci uložit, v našem případě */koncentrace_*, ke které přičteme číslo právě měřícího senzoru, takže nám vznikne */koncentrace_1,* dále */koncentrace_2* atp. Hodnotu, kterou posíláme, máme uloženou v poli *koncentrace*.

```
void loop(){
  for (int i = 0; i <= 5; i++) {
    MERENI_MQ(i);
    Firebase.setFloat(databaze, "/koncentrace_" + String(i + 1),
   koncentrace[i]);}
```

V neposlední řadě se hodí mít zálohu těchto dat. Proto na konci každého měření pošleme .json blok všech šesti naměřených hodnot.

Aby nám ale tato záloha k něčemu byla, je třeba ji nejprve opatřit datem a časem měření.

K tomu poslouží tři knihovny:

```
#include <NTPClient.h>
#include <WiFiUdp.h>
#include <TimeLib.h>
```

My si vytvoříme protokol a načteme klienta *hodiny*, který bude sbírat aktuální čas z evropského poolu, přičemž se odchýlíme o 3 600 sekund, tedy o právě 1 hodinu, čímž převedeme čas z GMT na náš český, středoevropský čas CET.

```
WiFiUDP protokol;
NTPClient hodiny(protokol, "europe.pool.ntp.org", 3600);
```

V bloku *void setup()* pak zahájíme klienta *hodiny*, podobně jako jsme tak učinili s Firebase databází či WiFi připojením.

```
  hodiny.begin();
```

V rámci *void loop()* bloku zaktualizujeme z online poolu *hodiny* - čas z těchto hodin naformátujeme a vepíšeme do textového řetězce *cas*. Dále deklarujeme funkci *time_t*, za pomoci které získáme tzv. Epoch čas, neboli UNIX čas, což je počet uplynulých sekund od data 1. 1. 1970 UTC. Tento epoch čas *t* nastavíme jako aktuální čas a vepíšeme ho do textového řetězce *den*, ve kterém ho převedeme na lidský čas, za pomoci funkcí *day()*, *month()* a *year()* pro den, měsíc, respektive rok.

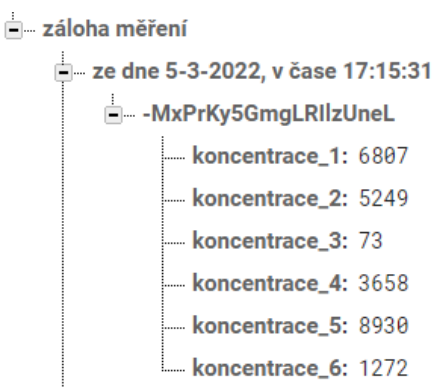
```
hodiny.update();
String cas = hodiny.getFormattedTime();
time_t t = hodiny.getEpochTime();
setTime(t);
String den = String(day())+ "-" + String(month())+ "-" +String(year());
```

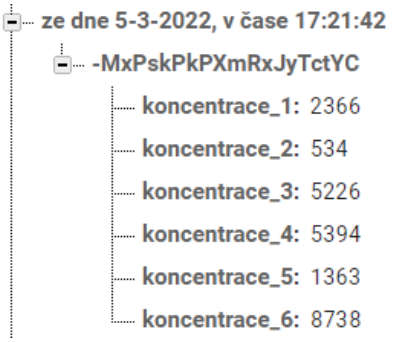
Nyní jsme připraveni ke zcela poslednímu kroku, poslání packetu dat do databáze. Tento balíček naformátujeme jako .json soubor, ve kterém vždy definujeme cestu k databázi a pak naměřenou hodnotu. Balíček pošleme funkcí *pushJSON()*, do jehož cesty uvedeme *den* a *cas* z předešlého kroku, abychom měli databázi přehlednou a mohli v ní jednoduše zpětně hledat.

```
String json = "{\"koncentrace_1\":" + String(koncentrace[0]) +
",\"koncentrace_2\":" + String(koncentrace[1]) + ",\"koncentrace_3\":" +
String(koncentrace[2]) + ",\"koncentrace_4\":" + String(koncentrace[3]) +
",\"koncentrace_5\":" + String(koncentrace[4]) + ",\"koncentrace_6\":" +
String(koncentrace[5]) + "}";

Firebase.pushJSON(databaze, "/záloha měření/ze dne " + den + ", v čase "
+ cas, json);
```

Výsledek zálohování pak může v databázi vypadat následovně:





VÝVOJ WEBOVÝCH STRÁNEK

Webová stránka má sloužit pouze jako hezké médium mezi daty měřených koncentrací.

Cílem bylo, aby se stránka sama aktualizovala, a to na místo jednoduchého, ale neefektivního obnovování celé stránky aktualizovala pouze data sbíraná z databáze, což se mi povedlo.

Z důvodu relativně špatné čitelnosti .html a hlavně velikého objemu (300+ řádků) kódu není zapotřebí uvádět konkrétní řešení, ale pouze uvedu, čím jsem se zabýval.

HTML

HTML tvoří kostru webové stránky.

V mém případě to byla tabulka 3x2, do jejíchž buněk jsem vždy dosadil po jednom grafu. Každému z těchto koláčových grafů jsem vytvořil popisek - jaké plyny měří a v jakých jednotkách je počítá. Zároveň jsem si nadefinoval <*span*>, kterému jsem přidělil originální identifikátor *id*, za které jsem později dosadil koncentraci, díky čemuž se pod grafem zobrazuje číselně přesná hodnota měřené koncentrace.

CSS

CSS tvoří styl, či vzhled stránky.

Inspiroval jsem se koláčovým grafem, kterému jsem upravil barvu, tloušťku, šířku, atp. Nastavil jsem si fonty a jejich velikost. Taktéž jsem vytvořil tmavé pozadí stránky a definoval pozice všech prvků. Dále jsem se inspiroval páčkou, které jsem poupravil animaci a barvu.

Také jsem si nadefinoval proměnné koncentrací, které jsem skripty později aktualizoval a používal pro spočítání procentuálního zastoupení pro intuitivní zobrazení koncentrací v grafu.

JavaScript

JavaScript zajišťuje funkčnost stránky.

Zde jsem vytvořil ze všeho nejdůležitější skript pro aktualizaci/synchronizaci dat s databází. Velmi jednoduše jsem na konec odkazu databáze pro konkrétní koncentraci přidal .json/, tedy například: https://martinznamenacekdb-default-rtdb.firebaseio.com/koncentrace_1.json/.

Takovýto odkaz lze přečíst i v normálním prohlížeči a zobrazí se pouze současná hodnota koncentrace, kterou jsem pak za pomoci funkce *get()* předal html části pro popisek a CSS části pro procentuální projekci výsledku.

Mimo to jsem ještě vytvořil tzv. *EventListener()*, který doslova naslouchá na event, čeká na funkci - kterou byla v mém případě změna frekvence aktualizace koncentrace - a jejímž spouštěčem byla jednoduchá páčka. Důvodem pro tuto funkci byla snaha zabránit eventuálnímu zahlcení databáze, ke kterému by mohlo dojít neustálou rychlou aktualizací dat.

Hostování

Po úspěšném vytvoření webových stránek už zbývalo jen přenést offline .html soubor na online doménu, k čemuž bylo zapotřebí hostování.

Hostování stránek jsem rovněž učinil přes Google Firebase, neb bylo nahrání stránek velice rychlé a hlavně nebylo opět potřeba nic platit. Google totiž umožňuje pro danou doménu zdarma až 10 GB uloženého statického obsahu a 10 GB sdíleného obsahu měsíčně.

V našem případě je problémový pouze sdílený obsah, protože se budou data stránky neustále aktualizovat, takže je třeba s tímto limitem počítat, ale při rozumných intervalech aktualizace dat (1000 ms a více) by to neměl být až takový problém.

Nejprve jsem si ve Firebase projektu vytvořil Webovou aplikaci a přiřadil jí doménu projektu.

Pro nahrání a zveřejnění stránek je zapotřebí nainstalovat již zmíněný Node.js příkazový řádek, a to včetně správce balíčků Npm a Chocolatey.

Po otevření Node.js příkazového řádku stačilo navigovat adresář za pomocí funkce *cd* do složky, ve které budeme všechny soubory potřebné ke zveřejnění stránky uchovávat.

V této složce jsem nainstaloval nástroje pro hostování stránek přes Firebase:

```
npm install -g firebase-tools
```

Dále je třeba se přihlásit do projektu přes Google účet a Firebase funkci autorizovat:

```
firebase login
```

Po úspěšném spojení s projektem můžeme projekt inicializovat, kdy zvolíme možnost pro hostování:

```
firebase init
```

Posledním krokem je nahrání stránek (přičemž se použijí soubory ze složky *public*, kam nahrajeme náš *index.html* soubor, který obsahuje veškerý zdrojový kód pro webové stránky):

```
firebase deploy --only hosting
```

Po tomto finálním kroku je stránka úspěšně online a projekt je tak dokončen.

POTENCIALITA VYUŽITÍ

Současně může mé měřicí zařízení fungovat jako orientační alarmový či kontrolní systém v místech, kde hrozí riziko zvýšení koncentrace mírně toxických plynů (oxid uhelnatý, oxid uhličitý, aj.) či úniků hořlavých substancí (methan, LPG, vodík, ethanol atp.).

Nesmírnou výhodou tohoto systému je fakt, že lze databázi propojit s dalším, zcela odlišným projektem a monitorovat takto více setů dat z různých měřičů. Například by šlo propojit měření z různých jiných projektů, či si takto třeba vytvořit zabezpečovací zařízení. Webové stránky lze totiž velmi jednoduše editovat a ekosystém, který jsem pro posílání a získávání dat vytvořil, je zcela univerzální a perfektně fungující.

Zařízení bohužel v současné době nelze osadit dalšími měřiči, poněvadž je použitá UNO WiFi deska limitována na 6 analogových pinů. Dalo by se však užít nové, totožné WiFi desky, a umožnit tak vzájemnou komunikaci těchto zařízení, anebo zakoupit nejvýkonnější desku Arduino Mega a k ní externě připojit WiFi modul. Tímto způsobem by bylo možné měřit za pomoci až 16 detektorů, měřičů a jiných senzorů, a ty pak na jednom místě sledovat.

ODKAZY

bundle nejdůležitějších odkazů (I, II, III na jednom místě)
https://linktr.ee/martinznamenacek

I. webové stránky ročníkové práce
https://martinznamenacek.web.app/

II. zdrojový kód - Arduino (.ino)
https://create.arduino.cc/editor/ard-platform-martinznamenacek/f22a9687-4f5f-476c-b765-e72264981666/preview

---

III. zdrojový kód - HTML kód (.html)
https://stackblitz.com/edit/web-platform-martinznamenacek?file=index.html

POUŽITÉ ZDROJE

- dokumentace k měřiči MQ-3
  https://www.sparkfun.com/datasheets/Sensors/MQ-3.pdf
- dokumentace k měřiči MQ-4
  https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-4.pdf
- dokumentace k měřiči MQ-5
  https://files.seeedstudio.com/wiki/Grove-Gas_Sensor/res/MQ-5.pdf
- dokumentace k měřiči MQ-7
  https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-7.pdf
- dokumentace k měřiči MQ-8
  https://datasheetspdf.com/pdf-file/904642/HANWEIELETRONICS/MQ-8/1
- dokumentace k měřiči MQ-135
  https://www.electronicoscaldas.com/datasheet/MQ-135_Hanwei.pdf
- dokumentace ke Google Firebase
  https://firebase.google.com/docs/guides

- informace o desce Arduino UNO WiFi Rev2
  https://www.hwkitchen.cz/arduino-uno-wifi-r2/

- návod na kalibraci a zahájení měření senzoru
  https://navody.dratek.cz/navody-k-produktum/senzor-plynu-mq-5.html
- návod na zprovoznění Firebase databáze s Arduino deskou
  https://tutorial.cytron.io/2020/11/25/send-data-to-firebase-using-arduino-uno-wifi-rev2/
- návod na zprovoznění Firebase hostování
  https://www.youtube.com/watch?v=w7xKZ5PWizs
- tipy pro debugging Firebase Arduino WiFiNINA knihovny
  https://github.com/mobizt/Firebase-Arduino-WiFiNINA/issues/18

- online program pro převedení zdrojového kódu z SCSS na CSS
  https://www.cssportal.com/scss-to-css/
- pomocník při vybírání barev pro lepší viditelnost prvků na webových stránkách
  https://imagecolorpicker.com/color-code/2596be
- inspirace pro polokruhové grafy
  https://codepen.io/vineethtrv/pen/xGjQOX
- inspirace pro páčku
  https://codepen.io/avstorm/pen/jOEpBLW