Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

# Projektna dokumentacija – Sistem za nadzor dimniških požarov

**Podatki o predmetu:**

| | |
|---|---|
| *Naziv predmeta:* | Projekt |
| *Študijski program:* | MAG Telekomunikacije, 2. letnik |
| *Nosilec predmeta:* | doc. dr. Boštjan Vlaovič, univ. dipl. inž. el. |
| *Asistent:* | mag. Ernest Gungl, mag. el. |
| *Študijsko leto:* | 2018/2019 |

**Študent:**

| | |
|---|---|
| *Vpisna številka:* | E5034453 |
| *Priimek in ime:* | Gaber Aljaž |
| *Datum:* | 17.1.2019 |

# KAZALO VSEBINE

# KAZALO SLIK

# 1 UVOD

V kurilni sezoni postanejo dimniški požari stalnica. Tako imajo z njimi dimnikarji in gasilci veliko dela. V povprečju v Sloveniji v dimnikih zagori 600-krat na leto. Vzrok za požare je navadno slabo vzdrževanje dimnika, včasih tudi slaba gradnja (npr. starejši sistemi brez toplotne zaščite). Število nepregledanih dimnikov prav tako narašča. Zelo problematični so dimniški sistemi s kaminom. Te večkrat vgradijo ''domači mojstri'' z veliko napakami, ki pa lahko imajo katastrofalne posledice. Sami požari so zelo težavni za gašenje, zato je ključno ustrezno in pravočasno posredovanje. V dimniku lahko tli več mesecev, požar pa izbruhne v nekaj sekundah. Zaradi teh dejstev se je porodila ideja za razvoj sistema, ki nadzoruje dimniške požare.

*Slika 1.1 – Dimniški požar*

# 2 **SPECIFIKACIJA ZAHTEV:**

## 2.1 Opis in namen sistema:

Znaki dimniškega požara so črn dim, prasketanje, bobneč zvok, smrad in stene dimnika so vroče. Sistem bi deloval tako, da se na dimnik namesti naprava, ki bi nadzirala temperaturo sten dimnika in merila gostoto dima (ppm). Če bi temperatura in gostota dima narastla preko optimalne vrednosti, bi se o tem obvestilo gasilce in rešilce. Sistem bi omogočal, reden nadzor dimnikov. V primeru požara pa bi bilo omogočeno pravočasno posredovanje.

Sistem bi bil namenjen za stanovanjske sisteme z dimniškimi sistemi. Uporabljali bi ga lahko na višjih in nižjih zgradbah. Ne bi pa bil primeren za industrijske objekte, saj so tam dimniški sistem drugačni od stanovanjskih. Prav tako industrijski objekti niso v tako veliki nevarnosti, zaradi strožjega nadzora dimnikov.

## 2.2 Izgled in delovanje sistema:

Sistem bi bil razdeljen na merilno in opozorilno enoto. Merilno enoto bi sestavljal mikrokrmilnik, s povezanim temperaturnim senzorjem bi merili temperaturo sten dimnika, z dimnim senzorjem pa gostoto dima. Merilna enota bi bila napajana preko baterije. Potrebna je dobra avtonomnost, saj je dostop do dimnika, kjer bi bila nameščena merilna enota po navadi težji. Pomembna je tudi zaščita pred zunanjimi elementi (npr. veter, dež, temperatura, sneg, … ). Torej bi merilno enoto dali v nepremočljivo ohišje in jo namestili v dimno kapo.

Ob izmerjenih vrednostih bi merilna enota vsake toliko časa poslala podatke na opozorilno enoto. Enoti bi komunicirali na večji razdalji in več merilnih enot bi komuniciralo z opozorilno enoto. Tam bi se podatki sprejeli in preko nje hranili na podatkovno bazo. Informacije s podatkovne baze pa bi obdelovali v omrežni aplikaciji, ki bi kronološko prikazovala vse zabeležene podatke. Ob izrednih vrednostih pa bi o tem obvestila reševalne službe. Opozorilna enota bi bila priključena na stalni vir napajanja, zato potrebe po avtonomnem delovanju na njej ne bi bilo.

## 2.3 Zahteve in ovire pri razvoju

1. Merilna enota mora biti odporna na zunanje elemente.

Delovati mora v vseh vremenskih pojavih in vseh letnih časih. Kljubovati, mora tudi vetru.

2. Merilna enota mora biti odporna na visoko temperaturo in dim.
3. Merilna enota mora  zaznavati vrednosti z veliko natančnostjo.

Ne sme se zgoditi, da se reševalne službe pokličejo zaradi napačno odčitane vrednosti.

4. Zanesljiva in varna komunikacija med merilno opozorilno enoto.

Dobra varnost sistema je nujna, saj lahko v primeru nezavarovanega sistema nepridiprav sproži alarm za požar.

5. Enkripcija poslanih podatkov.

Komunikacija mora biti med prenosom ustrezno zavarovana. Tako nepridiprav ne more spreminjati poslanih podatkov med samim prenosom.

6. Avtonomija delovanja.

Tukaj imamo v mislih predvsem merilno enoto, ki bo na strehi. Potrebno je daljše obdobje, brez menjave baterije, ker so strehe težko dostopne.

7. Zanesljiva in robustna komunikacija med opozorilno in merilno enoto.

Merilna enote bi bila nameščena na strehi.

8. Zavarovan fizični dostop do merilne in opozorilne enote.

Fizični dostop do naprave mora biti omejen, da do sistema ne more dostopati vsak.

9. Z gesli zavarovan dostop, do shranjenih podatkov in konfiguracijskih datotek omrežne aplikacije.

# 3 FUNKCIJSKA IN IZVEDBENA SPECIFIKACIJA

## 3.1 Konkurenca na trgu

Po proučevanju spletnih virov, nisem našel podobnega produkta, ki bi že bil na trgu. Trenutno obstajajo le protipožarne gasilne dimniške enote, ki ne vsebujejo nobene elektronike. Delujejo popolnoma na fizikalnih principih. Pri nadaljnjem razvoju projekta bi lahko vključili še gasilni del. Za zdaj pa se bomo osredotočili samo na prvotni načrt.

## 3.2 Znaki dimniškega požara

Za zaznavanje požara, smo se osredotočili na najbolj značilne in izrazite znake pri dimniškem požaru [1]:

- Glasno pokanje in bobnenje (nismo našli jakosti bobnenja v dB),
- naraščanje temperature in
- gost dim.

Vseh statističnih podatkov o dimniškem požaru nismo našli. Zato smo nekatere vrednosti vzeli iz statističnih podatkov o hišnih požarih:

- Povprečna temperatura hišnega požara je 593 °C ,
- gori več ur,
- tli več mesecev,
- gost požarni dim ima 270 ppm in
- glasnost je večja od 70 dB.

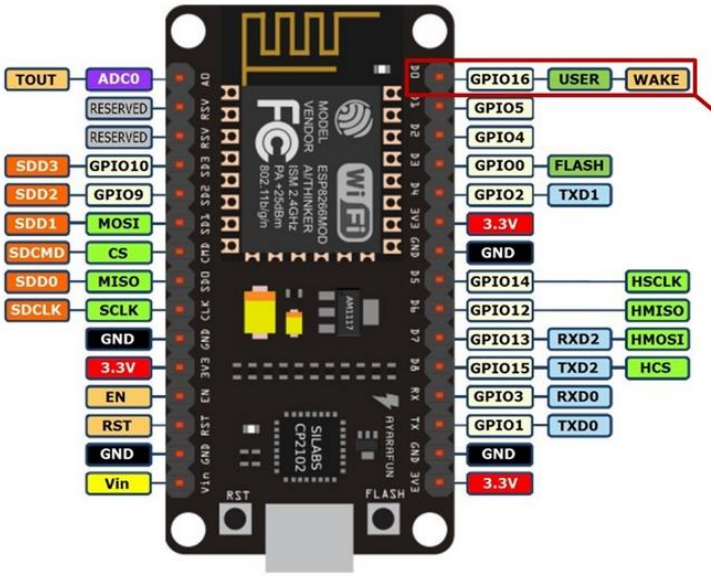
## 3.3 Lokacija opozorilne in merilne enote

Merilna enota bi bila nameščena v dimniško kapo. Tako bi bila zaščitena pred vetrom, ki ne bi mogel vplivati na meritve. Okoli naprave je potrebna izolacija, ki bi merilno enoto ščitila pred hlajenjem in segrevanjem dimniške kape v različnih letnih časih.

## 3.4 Izbira mikrokrmilnika

Za implementacijo obeh enot je potrebno uporabiti ustrezen mikrokrmilnik. Po premisleku smo kot končna kandidata izbrali  STM32L476 DISCOVERY in NodeMCU (ESP8266) .

### 3.4.1 NodeMCU (ESP8266)




*Slika 3.1 - NodeMCU*

Razlogi za uporabo:

- Nizka poraba (omogočanje avtonomnosti merilni enoti),
- integrirana Wi-Fi rešitev (ni potrebe po dodatnem WiFi modulu),
- podpira nekje do 7 povezanih komponent (v našem primeru imamo 3 vgrajene komponente, 2 senzorja in Lora modul),
- delovanje v temperaturah od -40 do 125 °C (dovolj za zaznavo požara),
- 32-bitni Tensilica Procesor, deluje na 160 MHz → posledično majhna poraba,
- WiFi sklad dopušča 80% procesorske moči za poganjanje kode,
- podpira širok nabor senzorjev in modulov,
- omogoča način globokega spanca (prihranitev energije),
- namenjen v veliki meri za IoT rešitve,
- razhroščevanje in nalaganje programske kode preko Micro USB priključka in
- veliko vodnikov ter podpore za razvoj aplikacij [2].

Slabosti:

- Potrebno ga je dokupiti in
- manj kvalitetna izdelava → posledično morebitne težave v delovanju.

Firmware :

Na voljo sta tudi dve najpogosteje uporabljene vrsti ''frimware-a'':

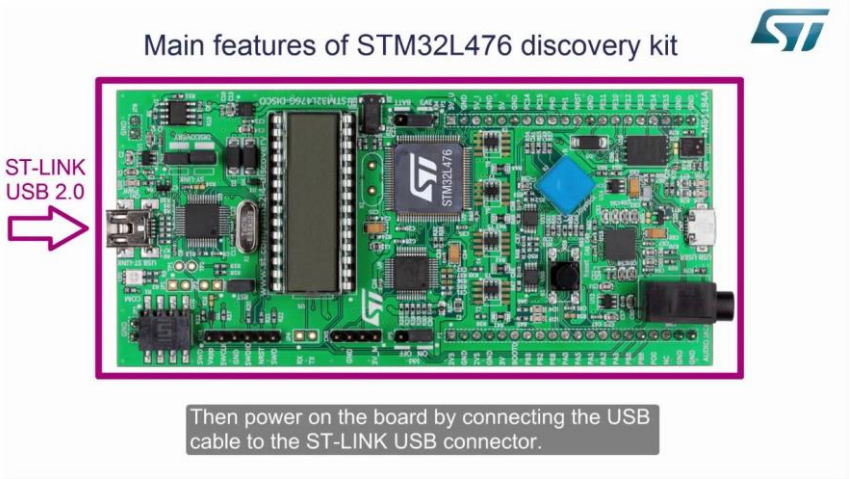
- NodeMCU

NodeMCU je odprokodni firmware, ki temelji na Lua programskem jeziku in uporablja SPIFFS datotečni sistem. NodeMCU je sprogramiran v programskem jeziku C. Je namensko razvit za ESP8266 [3].

- Firmware Arduino

Arduino je odprtokodna platforma. Je poceni, podpira jo veliko operacijskih sistemov, dobro programsko okolje, veliko knjižnic, velika podpora modulov. Uporabimo ga lahko tudi na NodeMCU, čeprav je ta zasnovan za Luo [3].

### 3.4.2 STM32L476




*Slika 3.2 - STM32L476*

Razlogi za njegovo kandidaturo pri izbiri so:

- Zmogljivo in hitro ARM jedro (Arm Cortex-M4 32-bit RISC),
- ima vgrajen 128 KB SRAM-a,
- deluje v razponu od -40 do +85/105/130 °C (visoke temperature pri požaru),
- majhna poraba (1.71 V - 3.6 V ''power supply'', pomembno zaradi avtonomnega napajanja opozorilne enote),
- USB povezava (razvoj in razhroščevanje),
- vgrajen temperaturni senzor (vgrajen tudi giroskop, gumbi, zaslon, ampak teh modulov ne potrebujemo pri našem projektu),
- na voljo je veliko razvojnih okolij (npr. Keil uVision),
- nanj lahko povežemo veliko senzorjev, zaradi velikega števila prostih ''pin-ov'' in
- na voljo so v laboratoriju [4].
-

Slabosti:

- Velika velikost,
- kot programski jezik lahko uporabimo samo C,
- ''firmware'' ni na voljo,
- malo vodnikov in literature in
- nima vgrajenega Wifi modula.

### 3.4.3 Končna izbira mikrokrmilnika

Za razvoj smo izbrali dva mikrokrmilnika NodeMCU. Najbolj tehtni razlogi za izbiro so:

- NodeMCU je primernejši za razvoj IoT storitev (kot je naš projekt),
- Že ima vgrajen WiFi modul, STM32L476 ga nima (enostavno povedano, na njem je lažje priključiti in omogočiti delovanje temperaturnega senzorja, kot pa komunikacijski modul WiFi),
- Na voljo je veliko vodnikov in literatue in
- Na voljo imamo dva ''firmware-a'' → Arudino ali Lua.

Za ''firmware'' smo izbrali Arudino, zaradi odprokodnosti, velike podpore merilnih modulov ter velike količine literature in knjižnic, ki so na voljo.

## 3.5 Pregled izbranih senzorjev

Po pregledu senzorjev smo prišli do naslednjega zaključka. Senzorja za dim in temperaturo smo izključno izbrali zaradi dobavljivosti in cene. Vsi ostali senzorji, ki merijo natančnejše parametre so veliko dražji in si jih je v prototipni fazi projekta tudi nesmiselno privoščiti (tukaj predvsem ciljamo na dimne senzorje). Senzorji se lahko nadgradijo naknadno.

### 3.5.1 Dimni senzor (MQ-2)


*Slika 3.3 - MQ-2*

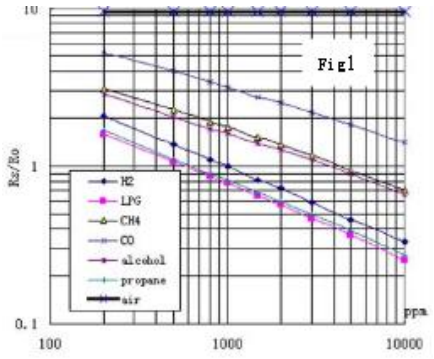
Prednosti:

- Dobra občutljivost (300 – 10000ppm),
- dolga življenjska doba,
- cena (3 $) in
- enostavno vezje (deluje tako, da se konduktivnost veča s koncentracijo dima oz. plina) [5].

Slabosti:

- Potrebna je kalibracija,
- poraba grelca <900mW (grelec je vedno prižgan),
- predhodno gretje 48 ur za optimalno delovanje in
- optimalno delovanje na 5V (na NodeMCU je na voljo samo 3.3V)

Dim je prisoten v vsakem dimniškem požaru. S tem senzorjem lahko zaznamo tako dim kot paro. Koncentracija dima pa mora biti med 10 – 1000 delci na milijon. Minimalna vrednost za zaznavo dima za požarne sisteme mora biti 10 ppm. Pri požaru, pa gost dim doseže okoli 270 ppm, kar pomeni, da je senzor primeren za uporabo v našem sistemu. MQ-2 ima naslednjo občutljivostno karakteristiko:




*Slika 3.4 - MQ-2 kalibracijska krivulja*

### 3.5.2 Temperaturni senzor (DHT-11)

*Slika 3.5 - DHT - 11*

Prednosti:

- Nizka cena (3$),
- majhna velikost,
- nizka poraba,
- deluje na 3.3 V ali 5V in
- avtomatska kalibracija.

Slabosti:

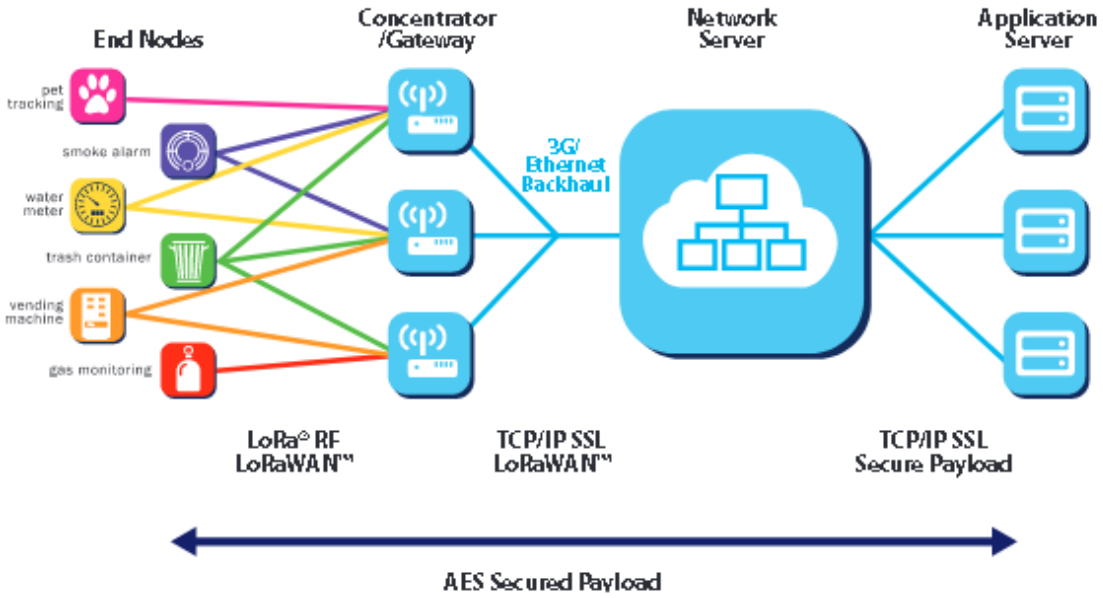
- Območje delovanja → temperature od 0-60 °C (za industrijsko implementacijo bi potrebovali večji temperaturni razpon, vsaj do 200 °C, a so takšni senzorji cenovno neugodni)

Temperature pri dimniškem požaru dosežejo do 600 °C. Takšne temperature lahko merijo zelo dragi senzorji. V našem primeru, je zaradi časovne omejitve glede nabave in prav tako cenovne bil izbran DHT-11 [6] .

## 3.6 Komuniacija opozorilne in merilne enote (LoRa)

Merilna in opozorilna enota komunicirata preko LoRaWAN omrežja. Ker gre za novejšo in manj znano tehnologijo jo bomo malce podrobneje predstavili. LoRaWAN je torej tehnologija za brezžično komunikacijo med baterijsko napajanimi napravami. Cilja predvsem na baterijsko napajane IoT naprave. Razvili so jo neprofitna družba LoRa-Alliance (kolaboracija podjetij ARM, RoyalTek, Gimasi, inmarsat, …). Zaradi kolaboracije več podjetij je LoRa globalni standard in ima omogočeno interoperabilnost z drugimi tehnologijami. Podpira jo 100 omrežnih operaterjev in se uporablja v več kot 100 državah na svetu.

Tehnologija je namenjena za uporabo v aplikacijah z nizko porabo energije, malimi prenosnimi hitrostmi in komunikacijo na dolge razdalje. Po navadi je LoRa omrežje sestavljeno iz centralnega vozlišča na katero je povezano več ostalih vozlišč. Centralno vozlišče služi kot ''gateway'' in je povezano z internetom. Vsako vozlišče komunicira s ''single hop'' brezžično povezavo. LoRa podpira dvosmerno komunikacijo in ''multicast'' [8]. Primer LoRa omrežja (tipično topologija izgleda takole):




*Slika 3.6 - Primer LoRa omrežja*

Komunikacija bazira na CSS (''chirp spread spectrum'') modulaciji, s katero je dodatno zmanjšana poraba energije. Komunikacija lahko poteka na različnih frekvencah in kanalih. Prenosne hitrosti pa so od 0,3 – 50 kbps [7]. Frekvence na katerih deluje LoRa komunikacija:

| | Europe | North America | China | Korea | Japan | India |
|---|---|---|---|---|---|---|
| **Frequency band** | 867-869MHz | 902-928MHz | 470-510MHz | 920-925MHz | 920-925MHz | 865-867MHz |
| **Channels** | 10 | 64 + 8 +8 | In definition by Technical Committee | In definition by Technical Committee | In definition by Technical Committee | In definition by Technical Committee |
| **Channel BW Up** | 125/250kHz | 125/500kHz | | | | |
| **Channel BW Dn** | 125kHz | 500kHz | | | | |
| **TX Power Up** | +14dBm | +20dBm typ (+30dBm allowed) | | | | |
| **TX Power Dn** | +14dBm | +27dBm | | | | |
| **SF Up** | 7-12 | 7-10 | | | | |
| **Data rate** | 250bps- 50kbps | 980bps-21.9kpbs | | | | |
| **Link Budget Up** | 155dB | 154dB | | | | |
| **Link Budget Dn** | 155dB | 157dB | | | | |

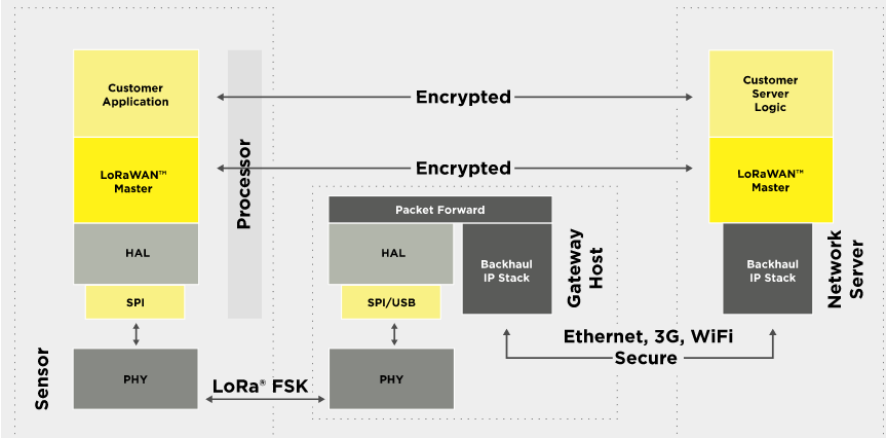
*Slika 3.7 - LoRa frekvenčno področje*

V Sloveniji LoRa deluje na frekvencah:

| | | |
|---|---|---|
| Slovenia | EU863-870<br>EU433 | CEPT Rec. 70-03 |

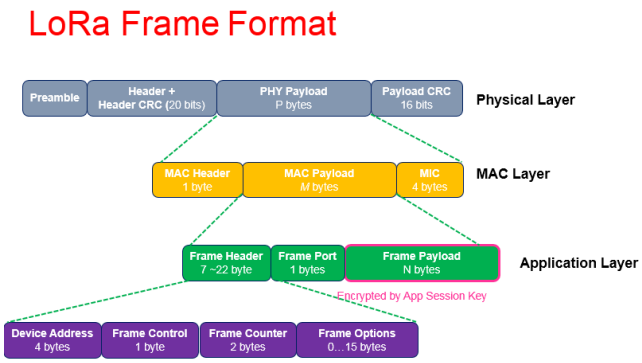
*Slika 3.8 - LoRa frekvenčni pas za Slovenijo*

Zaradi modulacije, frekvenc na katerih deluje in prenosnih hitrosti lahko en LoRa ''gateway'' pokriva površino več kvadratnih kilometrov. Med seboj pa se razlikujejo tudi končne naprave (razred A,B,C). Prav tako pa ima LoRaWAN definiranih več nivojev AES enkripcije. Primer LoRa komunikacije:




*Slika 3.9 - Primer LoRa komunikacije*

Sam LoRa paket z vsebujočo glavo pa izgleda takole :

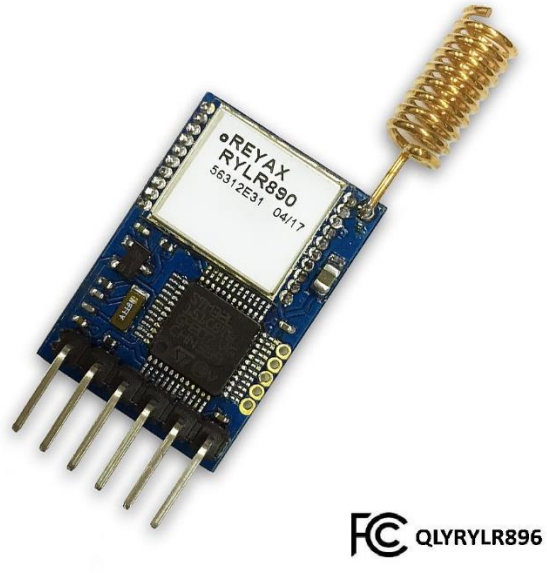



*Slika 3.10 - Primer LoRa paketa*

## 3.7 Izbira LoRa modulov:

Po prebranih specifikacijah in opisu vidimo, da je LoRa zelo primerna za uporabo v našem projektu. Za uporabo te tehnologija potrebujeta merilna in opozorilna enota, LoRaWAN modul. Zato smo naredili pregled nad moduli, ki so na voljo in v ožji izbor dali naslednje:

### 3.7.1 LoRa modul RYLR896




*Slika 3.11 - RYLR896*

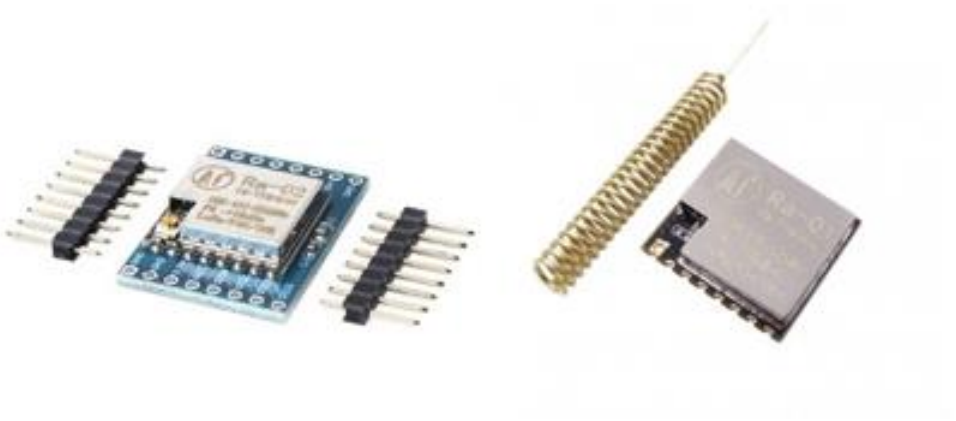
Razlogi za uporabo:

- Vsebuje čip SX1726,
- majhna poraba,
- visoka občutljivost,
- doseg do 8km,
- nadzor z AT ukazi,
- podprta AES128 enkripcija,
- Deluje na frekvencah 433 in 868 MHz ter
- integrirana PCB antena [9].

Slabosti:

- problem predstavlja cena, ki je 15 $ na enoto in
- dobava, ki je okoli enega meseca.

### 3.7.2 LoRa modul SX1278 (Ra-01/Ra-02)



*Slika 3.12 - Ra-01, Ra-02*

Razlogi za uporabo:

- Vsebuje čip SX1728,
- hitra dobava (teden dni),
- ''half-duplex'' SPI komunikacia,
- CRC pogon,
- podpira FSK, GFSK, MSK, GMSK,
- doseg do 5km,
- sleep mode: 0.2µA in
- deluje na 433 MHz [10].

Cena je pri tem modulu nižja. Na voljo sta dve verziji:

- verzija z nogami (5$; ne vsebuje antene; dobava okoli enega tedna)
- verzija brez nog (4$; vsebuje anteno, v tem primeru je še potrebno dobaviti še ESP12 – DIP adapter, a je potrebno pravilno zvezati modul na adapter)

Slabosti:

- Ne deluje na frekvenci 868 MHz, deluje samo za 433 MHz (tako se ne moremo povezati z že obstoječim omrežjem, ki je že v Sloveniji, to deluje 868 MHz),
- obstajata dve verziji modula vsaka s svojo slabostjo (prva verzija nima vključene antene, druga pa nima ''nog'' za priključitev na razvojno ploščico)

Končna izbira:

Po karakteristikah je v vseh aspektih boljši modul RYLR896. Ampak smo se vseeno morali odločiti za modul Ra-01, saj bi za RYLR896 dobava bila enostavno predolga. Funkcije Ra-01 se še podrobneje opisane v poglavju 5.7.

## 3.8 Shranjevanje podatkov na Firebase

Kot smo že povedali opozorilna enota deluje kot nekakšen ''sprejemni strežnik'', ki teče na mikrokrmilniku in sprejema izmerjene podatke z merilne enote. Ob sprejetju podatko se opozorilna enota poveže na lokalno WiFi omrežje in naloži sprejete informacije na FireBase, natančneje v podatkovno bazo Firebase.


*Slika 3.13 - Firebase*

Firebase je BaaS storitev, torej je sestavljena iz več sklopov: avtorizacija, shramba, avtorizacija, sporočanje v oblaku in podatkovna baza . V glavnem bomo uporabljali samo podatkovno bazo. Kot storitev s tem pridobimo vgrajeno varnost, možnost obdelave realno časovnih podatkov, hramba v oblaku, uporaba ''WebSocket-ov (namesto normalnega HTTP) … To pride predvsem v korist, za uporabo informacij v omrežni aplikaciji. Prav tako nimamo skrbi z infrastrukturo, saj Firebase ponuja vse v enem paketu [11].

## 3.9 SPLETNA APLIKACIJA

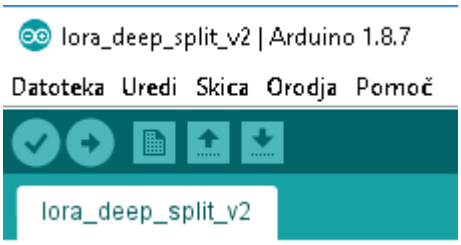
Omrežna aplikacija bi se izvajala na strežniku Apache, ki je zelo razširjen in je zanj na voljo ogromno dokumentacije. Razvoj bi potekal v PHP, s pomočjo ogrodja Codeigniter. Aplikacija bi tekla na operacijskem sistemu Windows 10. Vzpostavili bi jo v okolju XAMPP, ki že vsebuje Apache, MySql, PHP, … Podatke pa bi brali s podatkovne baze FireBase.


*Slika 3.14 - Apache*

Naša omrežna aplikacija bi prav tako, skrbela za obveščanje reševalnih služb preko programskega telefona. Zelo dober programski telefon je Ekiga, ki podpira SIP in VoIP.

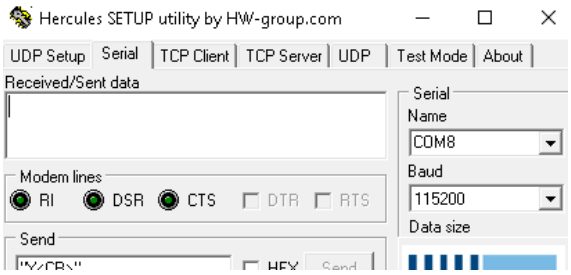
## 3.10 Razvojna in testna okolja

Ker smo izbrali ''firmware'' Arduino bomo naš projekt programirali v razvojnem okoljem Arduino IDE.



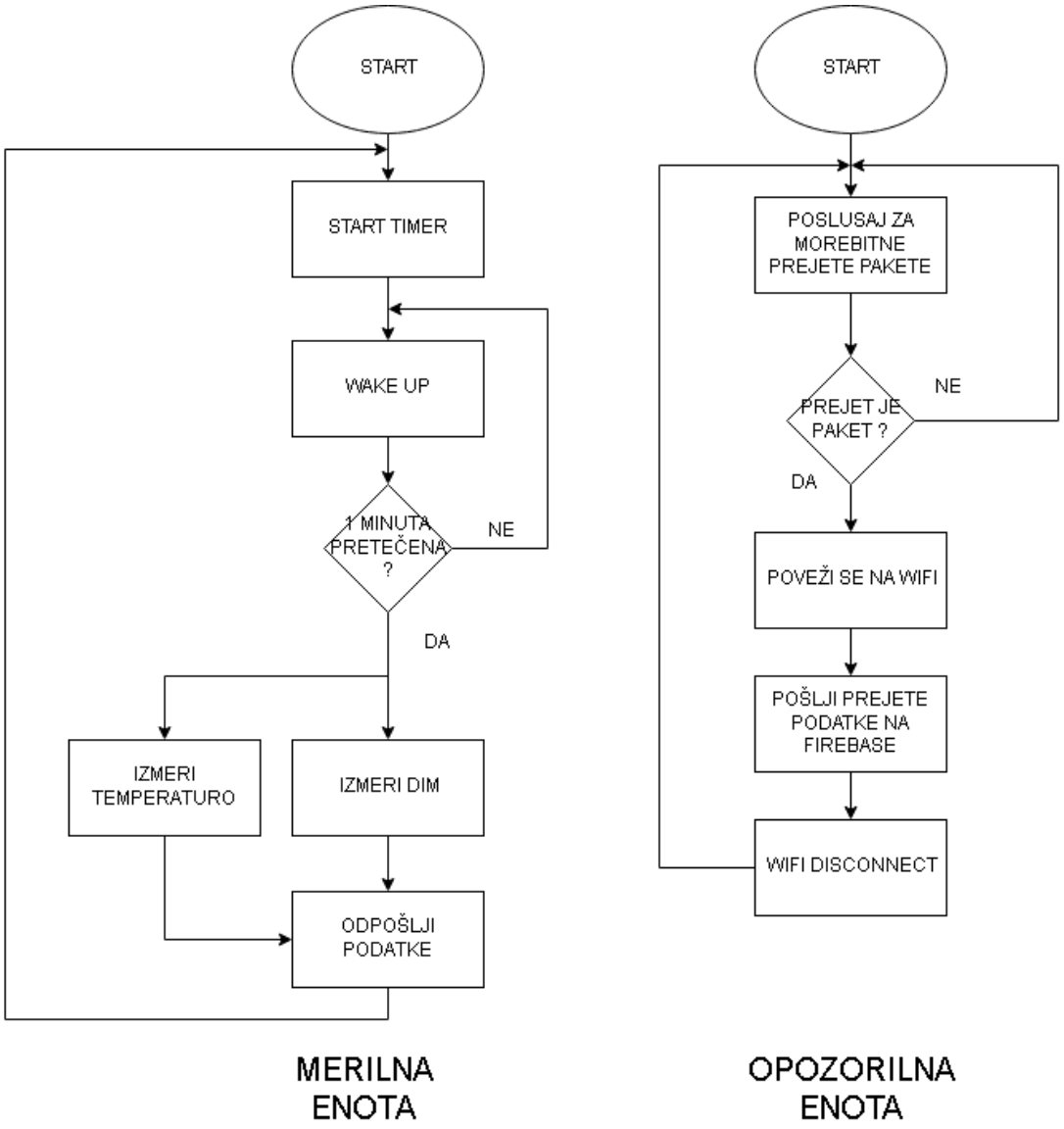

*Slika 3.15 - Arudino IDE*

Delovanje pa bomo nadzirali v programu Hercules (serijski vmesnik).




*Slika 3.16 - Hercules*

## 3.11 Poenostavljen princip delovanja sistema




*Slika 3.17 – Poenostavljen princip delovanja sistema*

## 3.12 Glavne funkcije našega sistema

1. Merjenje temperature sten dimnikov (merilna enota).
2. Merjenje gostote dima s pomočjo senzorja (merilna enota).
3. Proženje zvočnega alarma ob požaru (opozorilna enota).
4. Obvestitev reševalnih služb, če se je temperatura začela povečevati (omrežna aplikacija).
5. Vodenje časovne evidence, o preteklih podatkih (omrežna aplikacija).
6. Procesiranje podatkov in preverjanje njihove pristnosti (enkripcija podatkov na merilni in opozorilni enoti).
7. Shranjevanje sprejetih podatkov bi v podatkovno bazo (opozorilna enota).
8. Robustna komunikacija na dolge razdalje med opozorilno in merilno enoto (LoRaWAN).
9. Baterijsko napajanje opozorilne enote.
10. Majhna energijska poraba opozorilne enote in komunikacija na vsake toliko časa (globok spanec).

## 3.13 Predvideni terminski plan

- 10 ur → zasnova projekta
- 15 ur → izdelava funkcijske specifikacije in dobava modulov
- 40 ur → razvoj merilne enote
- 40 ur → razvoj opozorilne enote
- 30 ur → vzpostavljanje komunikacije med enotama
- 15 ur → testiranje in razhroščevanje
- 20 ur → pisanje dokumentacije

**Skupaj:170 ur**

# 4 IMPLEMENTACIJA PROJEKTA

## 4.1 Namestitev ''firmware-a''

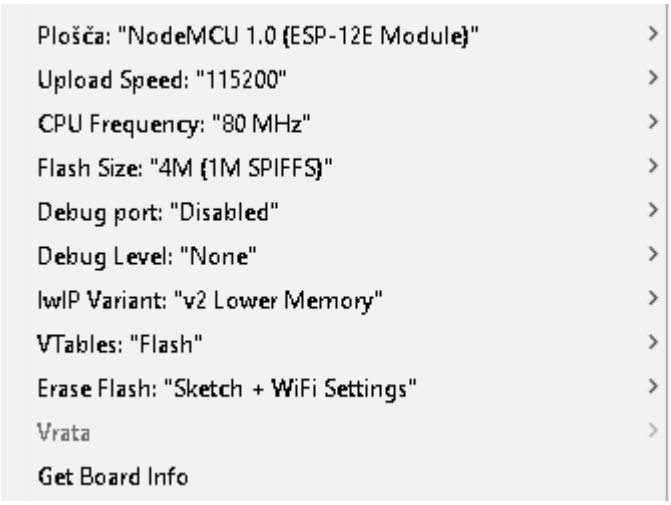
Kot smo že omenili na razvojnih ploščicah NodeMCU uporabljamo zapečeni program Arudino. Postopek za naložitev ''firmware-a'' na razvojno ploščico je enak za opozorilni in merilno enoto. To smo storili v Arduino IDE. Najprej smo prenesli Arduino Core za NodeMCU ESP-12. To je paket datotek v json formatu, ki vsebuje podporo za mikrokrmilnike z ESP8266 modulom.


*Slika 4.1 - Arduino Core*

Dodamo ESP8266 v seznam podprtih ploščic v razvojnem okolju. Nato gremo v Boards manager in prenesemo knjižnice za ESP8266.

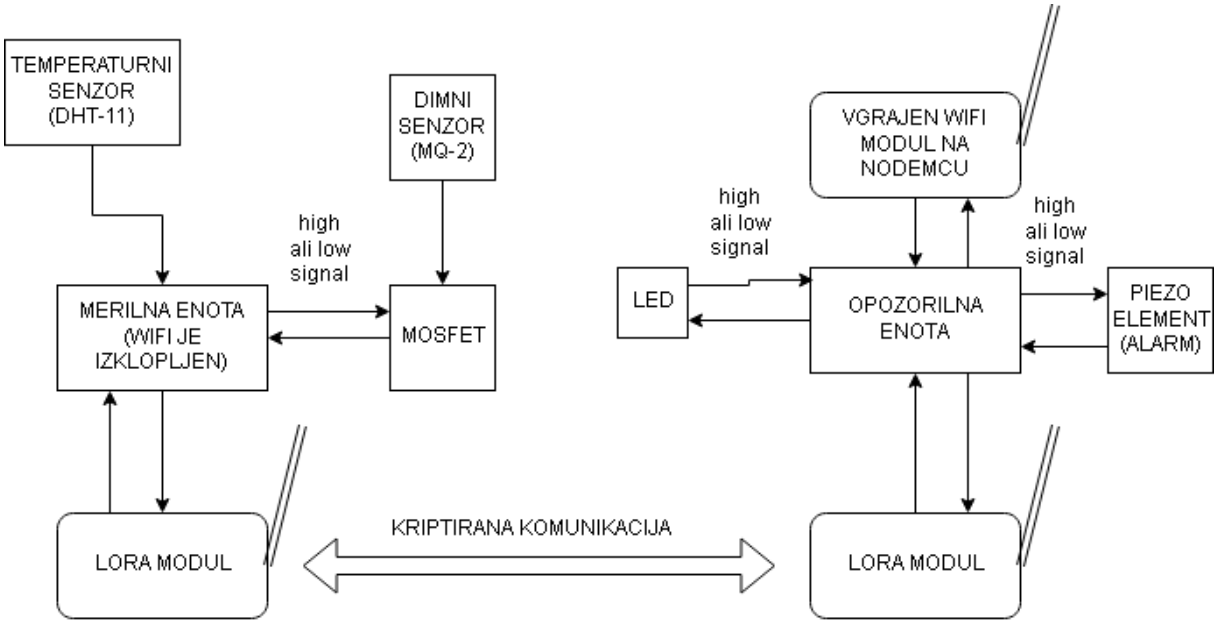

*Slika 4.2 - ESP8266 library*

Ko to storimo, se prikaže seznam ploščic z vgrajenim čipov ESP8266. V našem primeru izberemo ESP-12E modul. Ob tej izbiri tudi vedno zaganjamo in nalagamo programsko kodo na NodeMCU. Ob nalaganju je potrebno še izbrati ustrezna COM vrata:




*Slika 4.3 - COM port*

## 4.2 Izgled sistema:




*Slika 4.4 - Izgled sistema*

## 4.3 Implementacija merilne enote

Kot smo že omenili je merilna enota sestavljena iz:

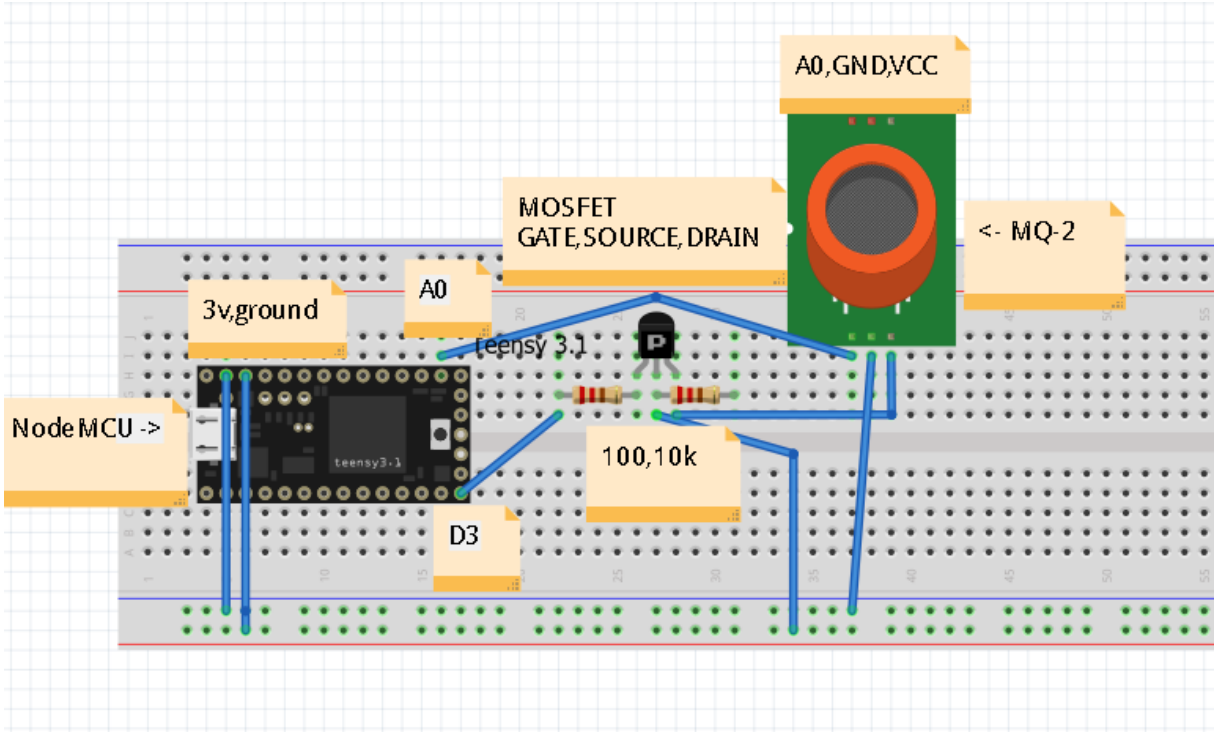
- Mikrokrmilnika NodeMCU,
- temperaturnega senzorja (DHT-11),
- dimnega senzorja (MQ-2) in
- LoRa modula (Ra-01).

### 4.3.1 Vezava senzorjev

Na merilni enota imamo povezana dva senzorja dimni senzor MQ-2 in temperaturni senzor DHT-11.

### 4.3.2 Vezava dimnega senzor MQ-2 (merilna enota)

Dimni senzor MQ-2 je analogni senzor. Vrednosti beremo preko analognega ''pin-a'' A0. Povzročil pa je tudi nekaj problemov. Pri merjenju porabe senzorja smo ugotovili, da se ta ne ugaša skupaj z NodeMCU. Tok je torej vedno tekel, kljub temu, da smo NodeMCU nastavili v globok spanec. Uporabili smo tudi MOSFET senzor, ki omogoča ugašanje grelnika, ki ga potrebuje MQ-2 za delovanje . Problem in njegova rešitev je podrobneje opisana v poglavju 4.5. Za povezavo potrebuje tri povezave, sama vezava pa izgleda takole:




*Slika 4.5 - Vezava dimnega senzorja*

Poenostavljen opis vezave:

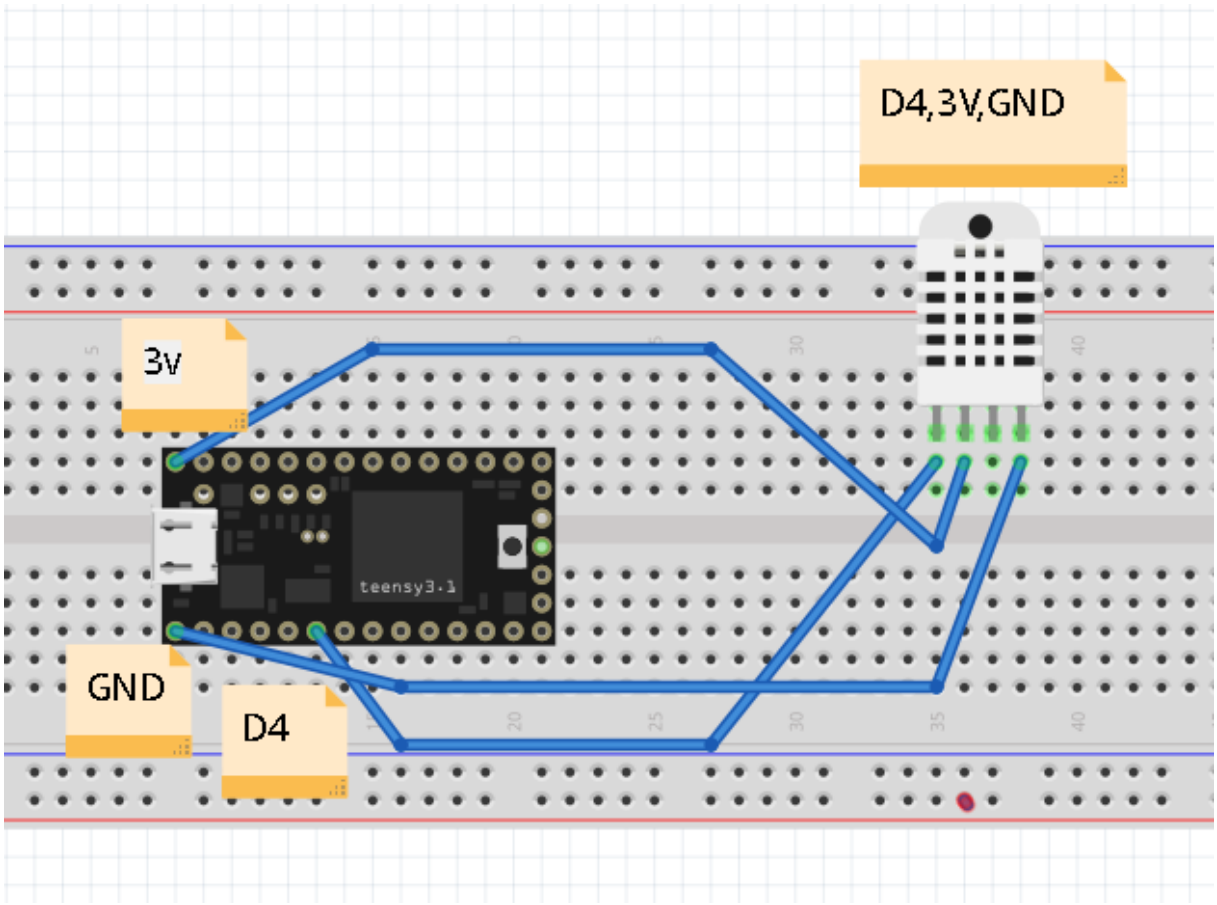
A0 (NodeMCU) <-> A0 (MQ-2)

GND (NodeMCU) <-> GND (MQ-2)

D3 (NodeMCU) <-> MOSFET <-> VCC (MQ-2)

### 4.3.3 Vezava temperaturnega senzorja DHT-11

DHT-11 smo povezali na naslednji način:




*Slika 4.6 - Vezava temperaturnega senzorja*

Poenostavljen opis vezave:

3V (NodeMCU) <-> 3V (DHT-11)

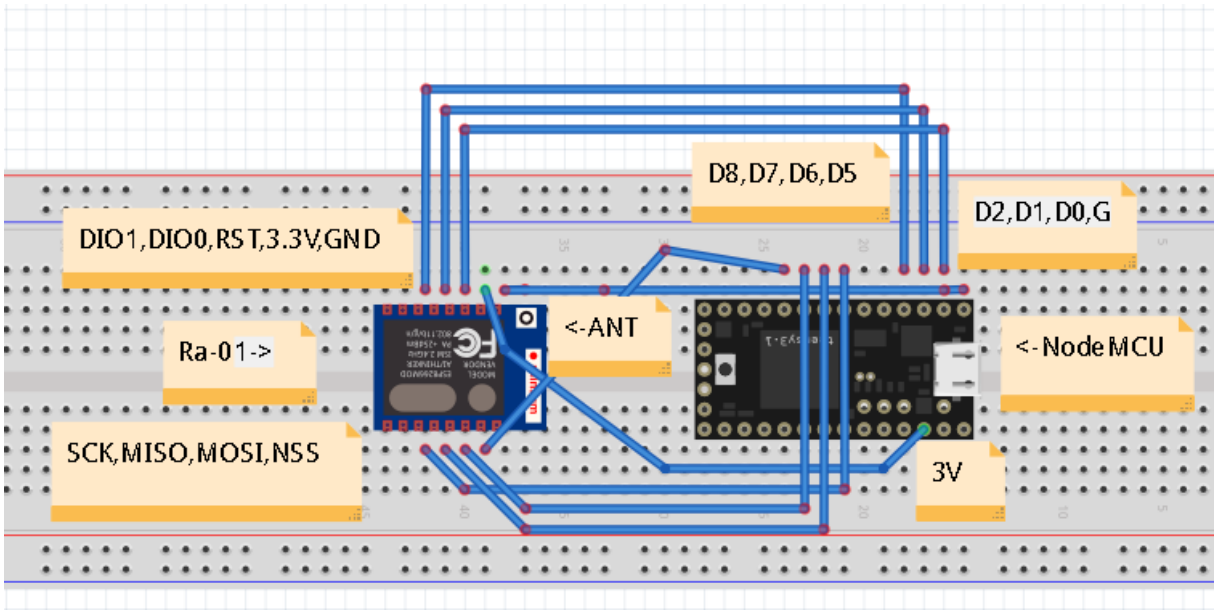
D4 (NodeMCU) <-> d4 (DHT-11)

GND (NodeMCU) <-> GND (DHT-11)

Ta senzor ni povzročal posebnih preglavic. Za delovanje senzorja se je uporablja že spisana knjižnica ''Adafruit DHT Humidity & Temperature  Sensor Library'' ( uporaba je podrobneje razložena v poglavju 5.1.

### 4.3.4 Vezava LoRa modula

Vezava LoRa modula se je izkazala za kar težavno nalogo in sicer zaradi velikega števila povezav. Potrebno je bilo natančno in potrpežljivo delo. Same povezave smo povezali s pomočjo vodnika [13]. Modul Ra-01 smo vezali na naslednji način:




*Slika 4.7 - Vezava LoRa modula*

Poenostavljen opis vezave:

Antena je nameščena na -> ANT

GND (Ra-01) <-> GND (NodeMCU)

DIO1 (Ra-01) <-> D2 (NodeMCU)

VCC (Ra-01) <-> 3.3V (NodeMCU)

DIO0 (Ra-01) <-> D1 (NodeMCU)

D0 (Ra-01) <-> RESET (NodeMCU)

SCK (Ra-01) <-> D5 (NodeMCU)

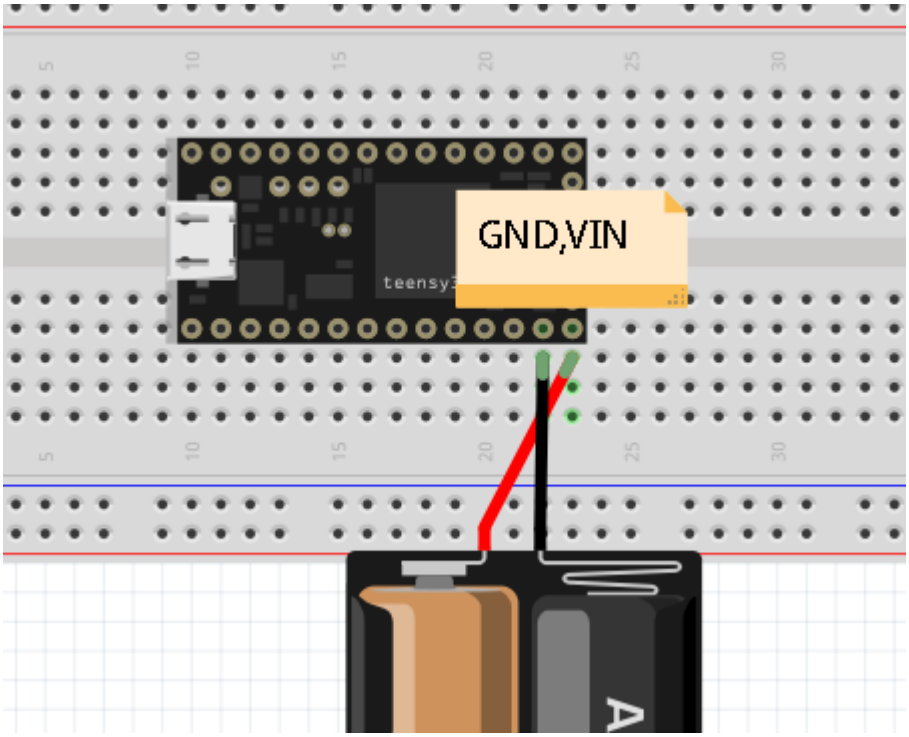
MISO (Ra-01) <-> D6 (NodeMCU)

MOSI (Ra-01) <-> D7 (NodeMCU)

NSS (Ra-01) <-> D8 (NodeMCU)

### 4.3.5 Poraba energije

Merilna enota je napajana preko baterijskega napajanja. Vezava z baterijo predstavlja sledeča slika:




*Slika 4.8 - Baterijska vezava*

Poenostavljen opis vezave:

+ →(Baterija) <-> VIN (NodeMCU)

-→(Baterija) <-> G (NodeMCU)

### 4.3.6 Merjenje porabe energije in omogočitev avtonomnosti:

Ker je merilna enota napajana preko baterije, je bilo potrebno razmisliti kako omogočiti avtonomnost enote (opozorilna enota je priključena na stalen vir energije, zato se na njej s tem problemom nismo ukvarjali). NodeMCU omogoča način globokega spanja, s katerim lahko omogočimo željeno avtonomnost merilne enote.

Pri ugotavljanju energijske porabe je bilo najprej potrebno izmeriti kakšen tok porablja NodeMCU v prižganem stanju, merjenju podatkov in pošiljanju preko LoRa modula. To smo storili s pomočjo USB merilnika ''Charge Doctor''. Ugotovili smo, da se v prižganem stanju uporablja tok nekje med 0,19 - 0,2 A.

Poizkus smo ponovili s testiranjem porabe energije v globokem spancu. Tako smo prišli do prvega problema. Opazili smo, da je tok v globokem spancu 0,13 A, moral pa bi biti okoli 0,2 – 0,5 uA. Razlog za ta problem je bil dimni senzor MQ-2. Na tem senzorju je nameščen grelnik, ki ga potrebuje za delovanje. Grelnik se kljub izklopu NodeMCU ni izključil (to smo videli po prižganih LED diodah na MQ-2). Ta problem smo rešili z vezavo MOSFET tranzistorja.

MOSFET je naprava, ki se kontrolira z napetostjo. Glavne komponente MOSFET-a so ''gate'', ''drain'' in ''source''. V primeru, da nanj priključimo napetost, ta generira električno polje, s katerim nadziramo tok med ''drain in ''source'', hkrati pa ni toka iz gate v MOSFET. To nam omogoča izklop in vklop MQ-2 (z visokim tokom (''HIGH'') izklopimo MOSFET → povzročimo električno polje, ki onemogoči napajanje MQ2 ; z nizkim tokom (''LOW'') vklopimo MOSFET → povzročimo električno polje, ki onemogoči napajanje MQ2) [15].

Ob implementaciji MOSFETA smo ponovno izvedli testiranje. Ugotovili smo, da smo z MOSFET-om uspešno izklopili MQ-2 senzor, saj je električni tok v globokem spancu enak 0 A (za meritev točnega toka bi potrebovali, natančnejši merilec napetosti).

### 4.3.7 Izračun porabe energije na merilni enoti

S testiranjem porabe merilne enote smo prišli do naslednjih podatkov:

- V času spanje je na merilni enoti el. tok enak: 0,2 – 0,5 uA (za točne rezultate bi potrebovali natančnejši merilnik, kot smo ga imeli trenutno na voljo, ta je kazal zgolj A na 3 decimalna mesta)
- V povprečju je merilna enota v prižganem stanju okoli 10 sekund in ima v tem času v povprečju električni tok 0,1321 A.

Porabo smo izračunali s pomočjo internetnega kalkulatorja [15], ki računa po naslednjih formulah:

C = C *0,85 → Kapaciteta baterije je zmanjšana za 15% zaradi faktorja samoizpraznitve.

t(zbujen) = t(budnosti) * št. vzbujenj

t (spanja na h) = 3 600 000 s – t (vzbujenja)

I (avg) = ((I(zbujen) * št. vzbujenj) + (I(spanje) * t (spanja na h)) / 1 ura

t (delovanja) = C / I (avg)

Rezultati pa so naslednji:

Primer št.1:

- napajanja z 3xAA (8100 mAh)
- 0,5 uA el. toka v času spanja
- 0,1321 A el. toka v budnem stanju
- ob 2 vzbujenjih na uro in
- času vzbujenja 10s

V tem primeru merilna enota avtonomno deluje **390,63 dni**.

Primer št.2:

- napajanja z 1xD (1200 mAh)
- 0,5 uA el. toka v času spanja
- 0,1321 A el. toka v budnem stanju
- ob 6 vzbujenjih na uro in
- času vzbujenja 10s

V tem primeru merilna enota avtonomno deluje **192,99 dni**.

Za naš projekt bi zadostovala oba primera, tudi primer št. 2, saj kurilna sezona ne traja več kot pol leta.

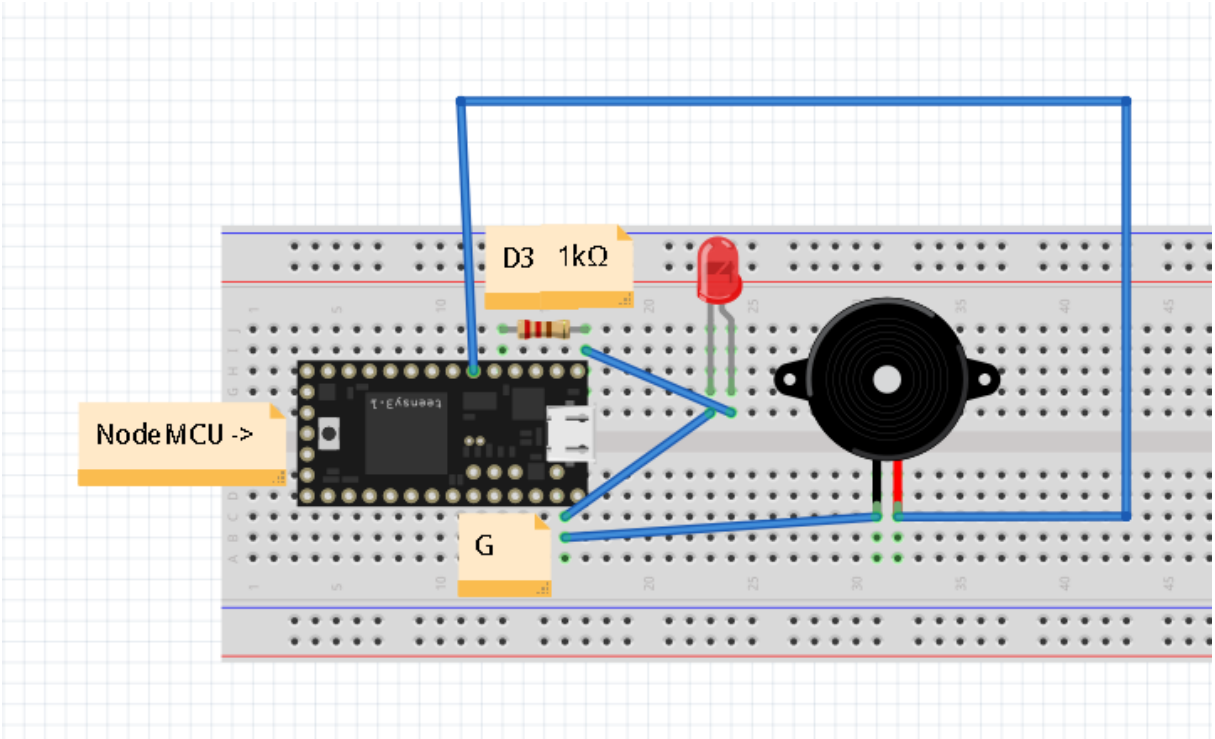
## 4.4 Implementacija opozorilne enote

Kot smo že omenili je merilna enota sestavljena iz:

- Mikrokrmilnika NodeMCU,
- LoRa modula (Ra-01),
- LED diode in
- piezo-elementa (piskača).

### 4.4.1 Vezava opozorilne enote

Na opozorilni enoti sta dodatno nameščeni še LED dioda, ki označuje prejet paket ter alarm, ki se vklopi ob preseženi optimalni vrednosti. Vezava izgleda tako:

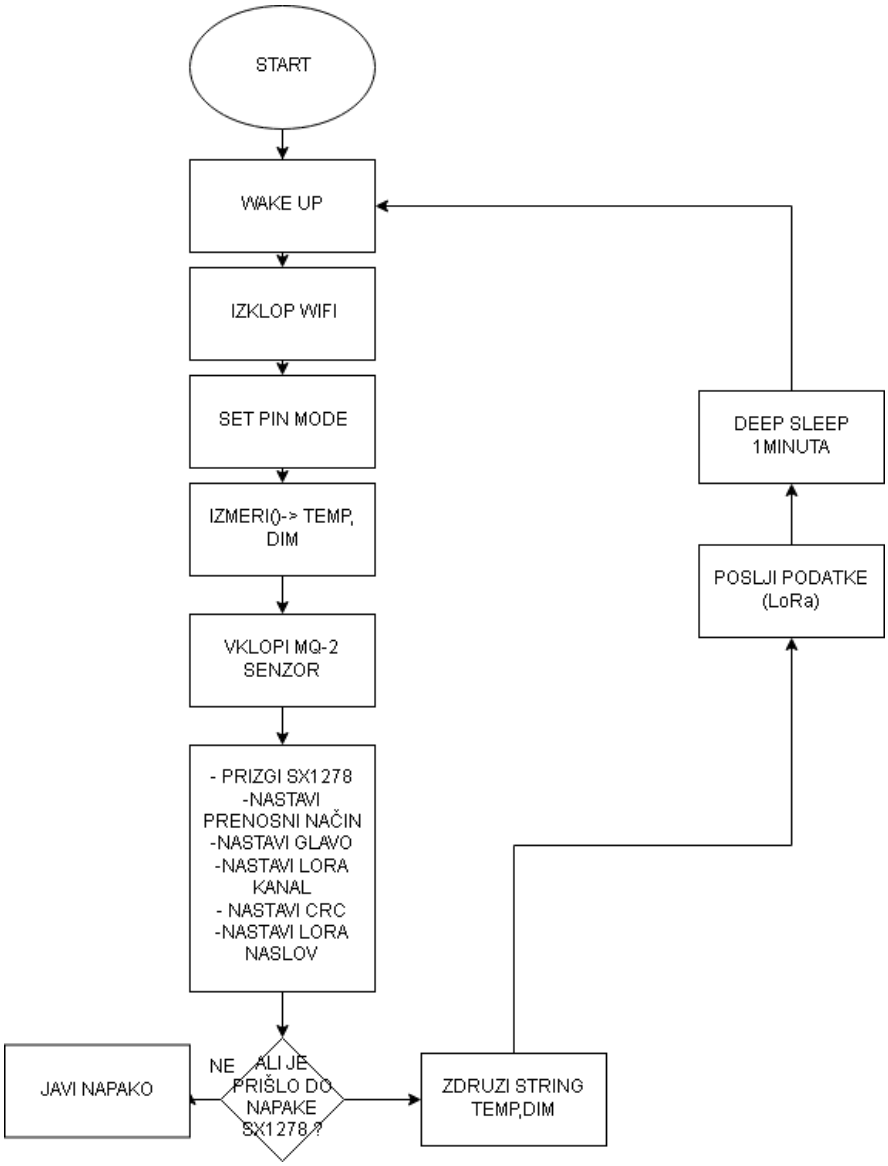



*Slika 4.9 - Vezava opozorilne enote*

### 4.4.2 Vezava LoRa modula na opozorilni enoti

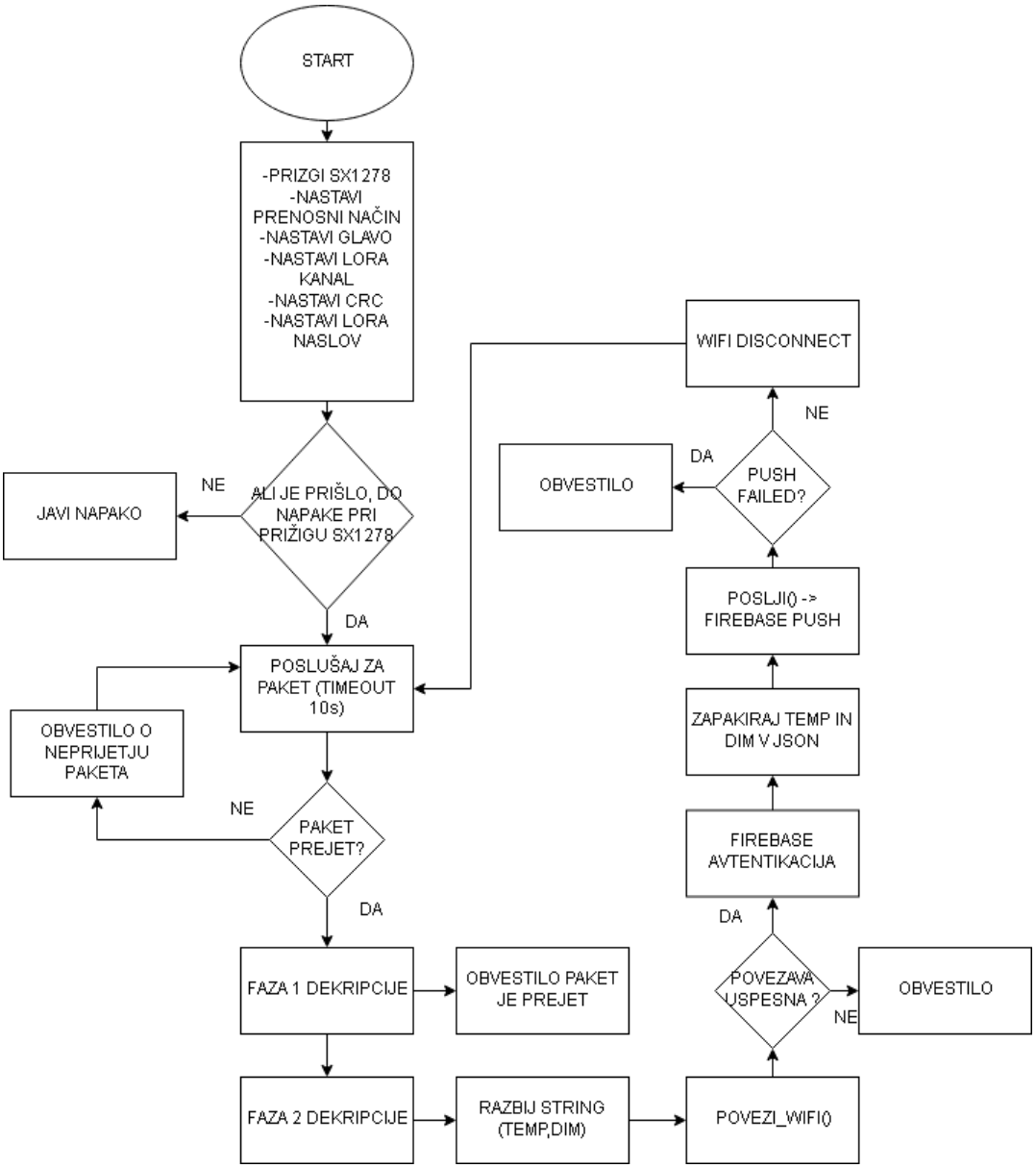
Vezava  modula je enaka, kot v poglavju 4.3.4.

## 4.5 Diagram poteka delovanje (merilna enota):




*Slika 4.10 - Diagram poteka, merilna enota*

## 4.6 Diagram poteka (opozorilna enota)




*Slika 4.11 - Diagram poteka, opozorilna enota*

# 5 IMPLEMENTACIJA POSAMEZNIH FUNKCIJ (PROGRAMSKI DEL)

Programska koda, za opozorilno enoto se nahaja v datoteki → lora_recieve_firebase_split_v3.ino
Programska koda, za merilno enoto se nahaja v datoteki → lora_deep_split_v2.ino

## 5.1 Merjenje temperature sten dimnikov (merilna enota)

Merjenje temperature sten dimnikov je izvedeno na merilni enoti, s funkcijo ''izmeri()'', ki je klicana v ''setup()''. Temperaturo izmerimo s senzorjem DHT-11, za katerega je potrebna knjižnica ''DHT-sensor-library'' [6], ki že vsebuje funkcijo za branje temperature s senzorja. Za uporabo je potrebno vključiti datoteko :

```
#include "DHT.h"
```

Potrebno je še definirati ''pin-e'' na temperaturnem senzorju ter tip senzorja:

```
#define DHTTYPE DHT11
#define dht_dpin 2

DHT dht(dht_dpin, DHTTYPE);
```

Pred odprtjem serijske komunikacija pa je potrebno še zagnati senzor z zgoraj definiranimi spremenljivkami:

```
dht.begin();
```

Temperaturo pa preberemo na naslednji način in jo shranimo v spremenljivko ''temp'', ki je celoštevilskega tipa:

```
temp = dht.readTemperature();
delay(500);

Serial.print("Temperatura = ");
Serial.print(temp);
Serial.print(" C  ");
delay(500);
```

## 5.2 Merjenje gostote dima s pomočjo senzorja (merilna enota)

Merjenje gostote dimnika, se izvaja na merilni enoti, s funkcijo ''izmeri()'', ki je klicana v ''setup()''. V poglavju 4.3.6 smo razložili, da moramo, za ustrezno porabo energije ugašati MQ-2 modul s pomočjo MOSFET-a. To storimo tako, da na začetku v funkciji ''setup'' definiran ''pin'' ''MQ2_Pin'' postavimo na nizko vrednost:

```
uint8_t MQ2_Pin = D3;

pinMode(MQ2_Pin, OUTPUT);
digitalWrite(MQ2_Pin,LOW);
```

Gostota dima se prebere z analognega ''pin-a'' (A0) in se shrani spremenljivko tipa ''float'':

```
dim = analogRead (analog_pin);
delay(500);
Serial.print("Dim = ");
Serial.print(dim);
Serial.print(" ppm");
```

Ob prehodu v merilne enote v globok spanec pa MQ-2 izklopimo, če ''MQ-2 pin'' postavimo na visoko vrednost:

```
digitalWrite(MQ2_Pin,HIGH);
```

## 5.3 Proženje alarma ob požaru (opozorilna enota)

Alarm sprožimo na opozorilni enoti, zgolj simbolično. Alarm je sestavljen iz LED diode in piezo elementa (piskača). Definirana sta s ''pin-oma'':

```
int led = 0; //D3
int buzzer = 2; //D4
```

Prožita se s funkcijama (petkratni pisk in petkratni utrip LED diode):

```
void alarm2 (){

  for(int i = 0 ; i<5;i++){
    digitalWrite(buzzer, HIGH);
    delay(100);
    digitalWrite(buzzer, LOW);
    delay(100);
  }
```

```
void alarm2 (){

  for(int i = 0 ; i<5;i++){
    digitalWrite(buzzer, HIGH);
    delay(100);
    digitalWrite(buzzer, LOW);
    delay(100);
  }
```

Funkciji se kličeta v primeru preseženih vrednosti dima in temperature znotraj funkcije ''loop()'':

```
if(dim > 800 || temp > 30){
    alarm1();
    alarm2();
}
```

## 5.4 Obvestitev reševalnih služb, če se je temperatura začela povečevati (omrežna aplikacija)

## 5.5 Vodenje časovne evidence, o preteklih podatkih (omrežna aplikacija)

Funkciji 5.4. in 5.5. nam ni uspelo implementirati, saj nam v času projekta, ni uspelo ustvariti še spletne aplikacije, katere zasnova je opisana v poglavju 3.8.

## 5.6 Procesiranje podatkov in preverjanje njihove pristnosti (šifriranje podatkov na merilni in opozorilni enoti)

Čeprav ima LoRa komunikacija s svojem delovanjem definirani šifriranje, tega modula naš Ra-01 ne podpira. Zato smo morali poiskati alternativno rešitev. Za šifriranje  na opozorilni in merilni enoti se uporablja kriptirni postopek ''xxtea'', ki je eno izmed novejših kriptografskih postopkov. Namenjeno je za implementacije v IoT aplikacijah [16]. Sam kriptirni algoritem izgleda takole:

```
#include <stdint.h>
#define DELTA 0x9e3779b9
#define MX (((z>>5^y<<2) + (y>>3^z<<4)) ^ ((sum^y) + (k[(p&3)^e] ^ z)));

void btea(uint32_t *v, int n, uint32_t const k[4]) {
  uint32_t y, z, sum;
  unsigned p, rounds, e;
  if (n > 1) {          /* Coding Part */
    rounds = 6 + 52/n;
    sum = 0;
    z = v[n-1];
    do {
      sum += DELTA;
      e = (sum >> 2) & 3;
      for (p=0; p<n-1; p++)
        y = v[p+1], z = v[p] += MX;
      y = v[0];
      z = v[n-1] += MX;
    } while (--rounds);
  } else if (n < -1) {  /* Decoding Part */
    n = -n;
    rounds = 6 + 52/n;
    sum = rounds*DELTA;
    y = v[0];
    do {
      e = (sum >> 2) & 3;
      for (p=n-1; p>0; p--)
        z = v[p-1], y = v[p] -= MX;
      z = v[n-1];
      y = v[0] -= MX;
    } while ((sum -= DELTA) != 0);
  }
}
```

*Slika 5.1 - Algoritem xxtea*

Prednosti uporabe so:

- En bit lahko spremeni celoten blok, tako ne moremo zaznati začetka sporočila,
- enostavnost uporabe,
- deluje samo v enem načinu in
- odporen je na več vrst napadov (npr. ''cut and join''). Samo identična sporočila vrnejo enak rezultat.

Njegova slabost pa je, da ne more prenašati zelo dolgih sporočil, kar pa v našem primeru (max. 53 32-bitnih besed).

Šifriranje implementirano s pomočjo knjižnice '' XXTEA Encryption and Decryption Library for Arudino'' [17]. Uvozimo jo z ukazom:

```
#include <xxtea-iot-crypt.h>
```

Merilna enota izvaja nalogo šifriranja, opozorilna pa nalogo dešifriranja. V našem primeru za dodatno varnost uporabljamo dvojno šifriranje.

Šifrianje poteka na naslednji način (merilna enota):
1.korak → Definiranje šifrirnih ključev.

```
String key1 = "1sdu7z!9x123";
String key2 = "11_ksL5PA13x";
```

2. korak → Nastavimo prvi šifrirni ključ (''key1'') in šifriramo željene podatke.

```
xxtea.setKey(key1);

// Perform Encryption on the Data
Serial.print(F(" Encrypted Data: "));
String result1 = xxtea.encrypt(skupaj);
result1.toLowerCase(); // (Optional)
Serial.println(result1);
```

3. korak → Nastavimo drugi šifrirni ključ (''key2'') in rezultat prejšnjega koraka.s

```
xxtea.setKey(key2);
String result2 = xxtea.encrypt(result1);
result2.toLowerCase(); // (Optional)
Serial.println(result2);

char * skupaj2 = new char [result2.length()+1];
strcpy (skupaj2, result2.c_str());
```

Dešifriranje poteka na naslednji način (opozorilna enota):

1.korak → Definiranje šifrirnih ključev.

```
String key1 = "1sdu7z!9x123";
String key2 = "11_ksL5PA13x";
```

2. korak → Nastavimo drugi šifrirni ključ (''key2'') in dešifriramo prejete podatke(ravno obratni postopek, kot na merilni enoti).

```
xxtea.setKey(key2);
Serial.println(F("1. faza dekripcije (key2)"));
String result3 = xxtea.decrypt(my_packet);
Serial.println(result3);
```

3. korak → Nastavimo drugi šifrirni ključ (''key1'') in vrnjen rezultat 2. koraka.

```
xxtea.setKey(key1);

Serial.println(F("2. faza dekripcije (key1)"));
String result4 = xxtea.decrypt(result3);
Serial.println(result4);
```

4. korak → Dešifrirani niz pretvorimo v polje tipa ''char''.

```
char * my_packet2 = new char [result4.length()+1];
strcpy (my_packet2, result4.c_str());

Serial.println("Po pretvorbi v char:/n");
Serial.println(my_packet2);
```

## 5.7 Robustna komunikacija na dolge razdalje med opozorilno in merilno enoto (LoRaWAN)

Kot smo že večkrat v dokumentaciji projekta omenili se komunikacija med merilno in opozorilno enoto izvaja preko LoRaWAN modula Ra-01 (opisan je v poglavju 3.7.2). Sedaj pa bomo opisali njegovo implementacijo.

Za njegovo delovanje smo uporabili knjižnico ''Lora Shield Arduino'' [18]. Prvi korak v kodi je uvoz knjižnice:

```
#include "SX1278.h"
```

Sledi nastavitev LoRa komunikacije po korakih:

1.Vklop modula.

```
if (sx1278.ON() == 0) {
  Serial.println("Power ON: OK ");
} else {
  Serial.println("Power ON: ERROR ");
}
```

2. Nastavljanje načina delovanja.

```
if (sx1278.setMode(LORA_MODE) == 0) {
  Serial.println("Nastavljanje načina: OK ");
} else {
  Serial.println("Nastavljanje nacina: ERROR ");
}
```

Tukaj smo tako na opozorilni, kt na sprejemni enoti nastavili način za prenos. Na obeh enotah je nastavljen način ''transmission''. Ob testiranju sem še poizkušal tako, da sem opozorilno enoto postavil v način ''recieve'', ampak na ta način komunikacija ni delovala. Tudi v priročniku je, kot primer na obeh enotah nastavljen na ''transmission'' in komunikacija deluje. To delovanje bi bilo še potrebno natančneje preveriti.

3. Omogočanje LoRa glave v paketu.

```
if (sx1278.setHeaderON() == 0) {
  Serial.println("Setting header ON: OK ");
} else {
  Serial.println("Setting Header ON: ERROR ");
}
```

4. Vklop CRC preverjanje napak za večjo robustnost komunikacije.

```
if (sx1278.setCRC_ON() == 0) {
  Serial.println("CRC ON: OK ");
} else {
  Serial.println("CRC ON: ERROR ");
}
```

5. Za čim nastavimo Ra-01 na največjo moč delovanja.

```
if (sx1278.setPower('M') == 0) {
  Serial.println("Nastavi moč: SUCCESS ");
} else {
  Serial.println("Nastavi moč: ERROR ");
}
```

6. Nastavljanje komunikacijskega kanala in naslova.

```
if (sx1278.setChannel(LORA_CHANNEL) == 0) {
  Serial.println("Nastavi kanal: OK ");
} else {
  Serial.println("Nastavi kanal: ERROR ");
}
```

Tukaj imamo na voljo več opcij. Na voljo je 24 različnih kanalov in 10 različnih pasovnih širin (7.8 kHz, 10.4 kHz, 15.6 kHz, 20.8 kHz, 32.1 kHz, 41.7 kHz, 62.5 kHz, 125 kHz, 250 kHz, 500 kHz). Pasovna širina je tudi najboljši način, s katerim lahko razpolagamo kakšen bo naš doseg signala. Pogoj za komunikacijo je, da merilna in opozorilna enota poznata naslova druge enote. Naslov nastavimo na naslednji način:

```
if (sx1278.setNodeAddress(LORA_ADDRESS) == 0) {
  Serial.println("Nastavljanje naslova: OK ");
} else {
  Serial.println("Nastavljanje naslova: ERROR ");
}
```

Izbira prenosnega kanala praktično ne vpliva na samo delovanje. S kanali samo nastavimo, katera je centralna frekvenca delovanja. Njihov razpon je od 433,086 MHz do 434,754 MHz. Pomembnejša je pa nastavitev pasovne širine. Z večjo pasovno širino lahko prenesemo večjo količino podatkov, ampak s tem zmanjšamo doseg signala. Z manjšo pasovno širino pa lahko prenesemo manjšo količino podatkov, a na veliko večji razdalji. Testirali smo pasovne širine 125 kHz, 250 kHz in 500 kHz, saj prenašamo dokaj veliko količino podatkov. Ko sta izmerjena temperatura na merilni in opozorilni enoti izmerjeni, se ti dve zašifrirata in nastane črkovni niz dolžine približno 25 črk. Sami rezultati testiranja so podrobneje opisani v poglavju 5.7.1.

7. Oddaja paketa (merilna enota)

Pri oddaji moramo nastaviti pravilni naslov opozorilne enote in poslati paket na naslednji način:

```
e = sx1278.sendPacketTimeout(LORA_SEND_TO_ADDRESS, skupaj2);
Serial.print("Packet sent, state ");
Serial.println(e, DEC);
```

Podatke pošljemo v obliki : ''12,800''. Razlog za ta način je, da lahko izmerjeno vrednost gostote dima in temperature pošljemo znotraj enega paketa. To naredimo s ''parsingom''.

8. Sprejem paketa (opozorilna enota)

Najprej nastavimo časovnik za prejem sporočila:

```
e = sx1278.receivePacketTimeout(10000);
```

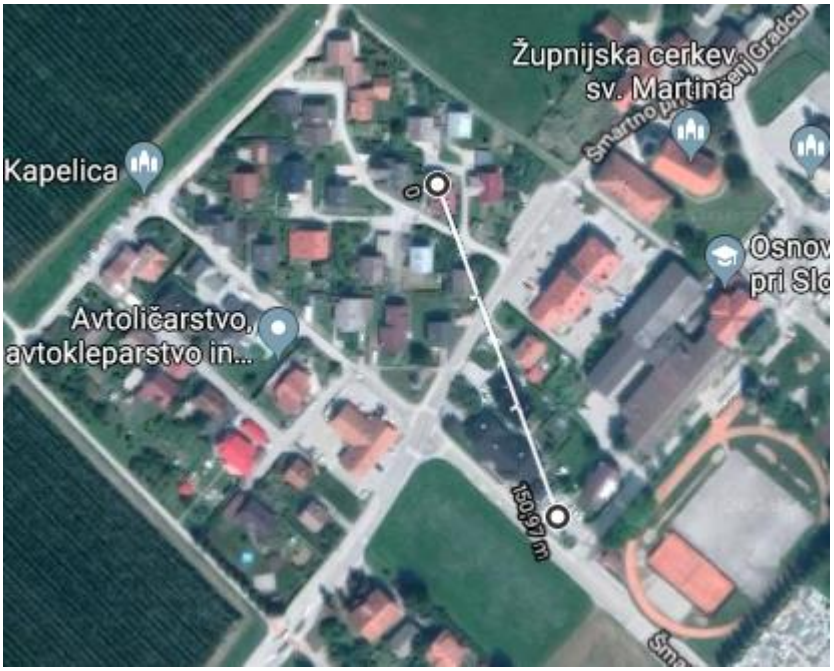
V primeru, da časovnik poteče nas modul po serijskem izhodu obvesti, da ni bilo prejetega paketa in časovnik ponovno zaženemo. V primeru prejetega paketa pa ga preberemo in shranimo v črkovni niz:

```
for (unsigned int i = 0; i < sx1278.packet_received.length; i++) {
  my_packet[i] = (char)sx1278.packet_received.data[i];
}
```

### 5.7.1 Testiranje dosega LoRa modula (Ra-01)

Doseg našega modula sem testiral tako, da ob prejetju paketa na opozorilni enoti, na piskaču pustim zvočni signal (poglavje 4.4.1). Torej premikal sem samo opozorilno enoto, ki sem jo priključili na

baterijsko napajanje. Doseg sem testiral v domačem kraju, ki je lep primer gosto naseljenega območja. S pasovno širino 500 kHz in 250 kHz sem dosegel doseg med enotama okoli 150-200m, kar je prikazano na naslednji sliki (razdalja je izmerjena od lokacije namestitve merilne postaje do točke zaznanega signala):




*Slika 5.2 - Doseg z BW 500 kHz, 250 kHz*

Boljše rezultate pa smo dosegli pri pasovni širini 125 kHz. Signal je segel približno 610m (razdalja je izmerjena od lokacije namestitve merilne postaje do točke zaznanega signala):


*Slika 5.3 - Doseg z BW 126 kHz*

S takšnim dosegom bi lahko lepo pokrili moje domače naselje. Na eno opozorilno enoto, bi lahko v tem območju teoretično pošiljali 24 različnih merilnih enot (na voljo imamo 24 različnih kanalov na Ra-01). Signal bi še lahko izboljšali z manjšanjem pasovne širine, a nam je za dodatna testiranja zmanjkalo časa.

## 5.8 Shranjevanje sprejetih podatkov v podatkovno bazo (opozorilna enota)

čas

Za shranjevanje prejetih podatkov, smo uporabili spletno storitev Firebase (poglavje 3.8). To funkcionalnost smo implementirali samo na opozorilni enoti. Najprej je bilo potrebno inicializirati Firebase odjemalca in podati avtorizacijsko kodo, ki omogoča pošiljanje prejetih podatkov na podatkovno bazo.

```
#define FIREBASE_HOST "nodemcu-8b28d.firebaseio.com"
#define FIREBASE_AUTH "qvA1SoHl9NRZIiuBVTO6Sgv7jQEKDMmWkE1apcfd"
```

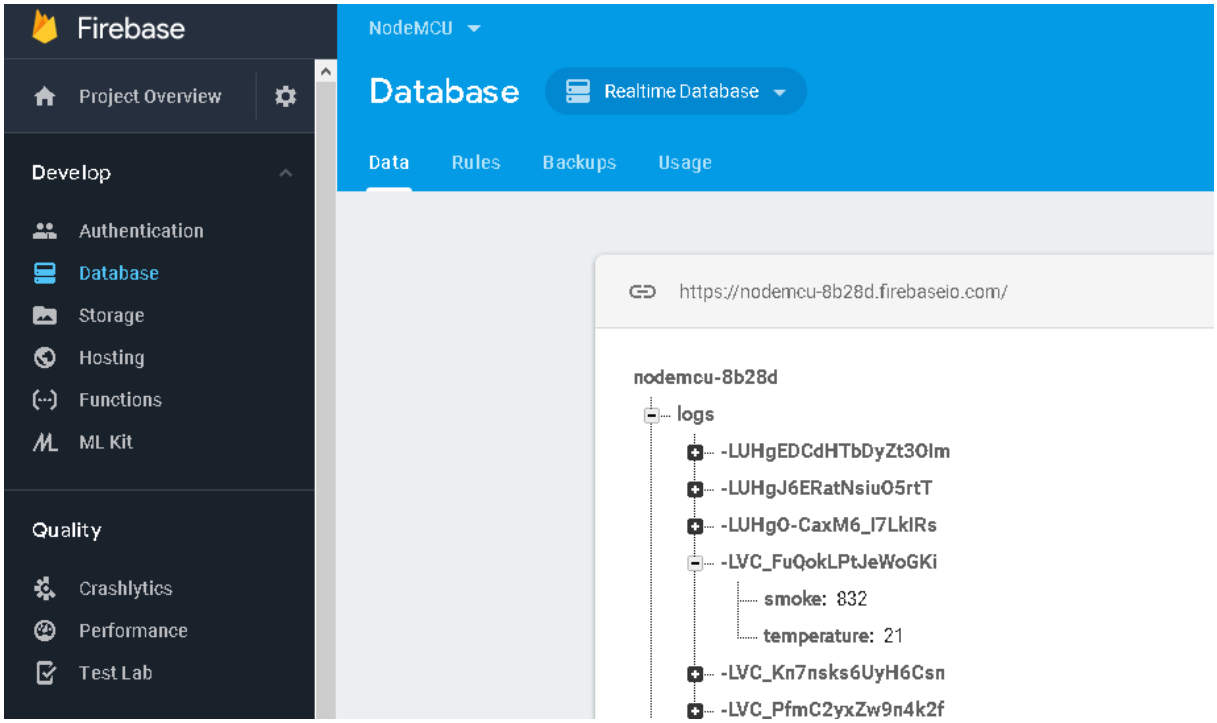
Ob prejetju paketa z merilne enote, se opozorilna enota poveže na lokalno WLAN omrežje. V primeru nedosegljivosti omrežja poteče časovnik in se postopek pošiljanja podatkov na Firebase prekine. V primeru uspešne povezave na WLAN omrežje se avtorizira Firebase odjemalca:

```
Serial.println("Firebase aventikacija...");
Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
```

Prejete podatke nato zapakiramo v JSON format . Primer: {''temp':,''25'',''smoke'':''810''}. Ter pošljemo na Firebase:

```
JsonObject& root = jsonBuffer.createObject();
root["temperature"] = temp;
root["smoke"] = dim;
String name = Firebase.push("logs", root);
// handle error
if (Firebase.failed()) {
Serial.print("pushing /logs failed:");
Serial.println(Firebase.error());
return;
}
Serial.print("pushed: /logs/");
Serial.println(name);
```

Ko se postopek pošiljanja zaključi, prekinemo povezavo z WiFi omrežjem, saj tako zagotovimo, da se v morebitnem izpadu WLAN omrežja, ob njegovi ponovni vzpostavitvi lahko ponovno povežemo nanj. Naložene podatke pa lahko na internetni strani storitve Firebase v tej obliki:



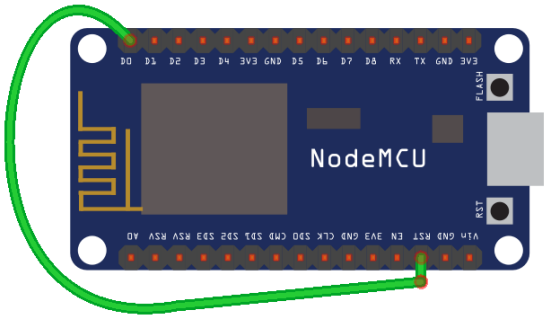

*Slika 5.4 - Firebase shranjenvanje*

## 5.9 Baterijsko napajanje opozorilne enote

Baterijsko napajanje je izvedeno z vezavo, ki je opisana v poglavju 4.3.3.

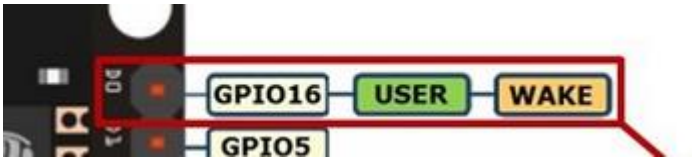
## 5.10 Majhna energijska poraba opozorilne enote in komunikacija na vsake toliko časa (globok spanec).

NodeMCU omogoča tudi način globokega spanca. Najprej je potrebno povezati omogočiti povezavo ''RST <-> D0'':




*Slika 5.5 - Vezava za globok spanec*

Če pogledamo NodeMCU ''pinout'' vidimo, da ima NodeMCU na GPIO16 posebno funkcije ''WAKE'':




*Slika 5.6 - GPIO16 -> WAKE*

Globok spanec deluje tako, da je pin ''RST'' vedno nastavljen na visoko vrednost (''HIGH''), ko je mikrokrmilnik prižgan. Kadar pa ''RST'' prejme nizko vrednost (''LOW'') se mikrokrmilnik resetira. Nastavimo pa lahko tudi časovnik, ki zbudi NodeMCU vedno, ko časovnik poteče. Časovnik nastavimo z že implementiran v funkcijii ''ESP.deepSleep()'' na znotraj ''setup()'':

```
Serial.println("Deep sleep 30s ...");
ESP.deepSleep(30e6);
delay(500);
```

# 6 STROŠKI DELA

Skupni stroški dela:

- 1x NodeMCU (9€)
- 1x NodeMCU (izposojen)
- 1x MOSFET (izposojen)
- 1x DHT-11 (3€)
- 1x MQ-2 (3€)
- 30 žic moški-moški, moški-ženska (4,5€)
- 2x Ra-01 LoRa (15€)
- 1x ESP 12-E adapter (3€)
- 2x antena za Ra-02 (1 €, neuporabljena)

**Skupni strošek izdelave → 37,5 €**

Komentar:

Cena projekta, se je izkazala, kot malce visoka. Ceno bi se dalo zmanjšati z nabavo komponent s Kitajske, ampak potem nebi imeli zagotovila, glede prejetja komponent.

# 7 DEJANSKA PORABA UR

Poraba ur za razvoj:

- 30 ur → zasnova projekta
- 10 ur → izdelava funkcijske specifikacije
- 30 ur → razvoj merilne enote
- 25 ur → razvoj opozorilne enote
- 10 ur → vzpostavljanje komunikacije med enotama
- 10 ur → testiranje in razhroščevanje
- 20 ur → izdelava dokumentacije

**Skupni porabljeni čas → 135 ur**

Komentar:

Porabljen čas se je izkazal, da ga je bilo malce manj od predvidenega (celotne porabe si nismo beležili, zato so lahko tudi odstopanja od realne porabe časa). Ni se pa pričakovalo tolikšne porabe časa pri sami zasnovi projekta in pri izdelavi dokumentacije.

# 8 NEDOSEŽENI CILJI

Pri projektu, večinoma zaradi časovne omejitve pri izdelavi projekta nismo izvedli naslednjih ciljev:

- Kalibracija dimnega senzorja (MQ-2),

Sama kalibracija bi enostavno vzela preveč časa, saj za njo ni nobenih navodil. Na spletu se je našel zgolj eno že implementirano rešitev izvedeno s programsko kodo, ki pa je daljša od naše spisane kode in kompleksna za razumevanje.

- izdelava spletne aplikacije in
- intenzivnejše testiranje dosega (doseg signala za manjše pasovne širine).

# 9 NADALJNJE DELO

Seznam idej za nadaljnje delo:

- Implementacija komunikacije 868MHz (potrebovali bi drug modul npr. RYLR896, ki je opisan v poglavju 3.7.1),
- testiranje dosega,
- optimiziranje delovanja, zmanjšati čas budnosti merilne enote (posledično bi zmanjšali porabo),
- spletna aplikacija,
- izdelava ohišja,
- uporaba natančnejših senzorjev in

Sedaj uporabljeni senzorji se niso izkazali za najbolj primerne. Potrebno bi bilo uporabiti dimni senzor, katerega kalbracija je standardiziran postopek in je hkrati natačnejši. Temperaturni senzor pa bi moral delovati v večjem temperaturnem območju (vsaj do 200 °C).

- izdelava spletne aplikacije (opis in zasnova v poglavju 3.9).

# 10 SKLEP

Ta projekt je bil popolnoma nova izkušnja, saj se v dosedanji študijski poti še nismo soočili z izvajanjem projekta, ki ga zasnujemo sami. Z delom na njem smo dobili vpogled v izvajanje projekta in na katere stvari moramo pri njem paziti. Pri tem projektu smo se naučili veliko koristnih in novih znanj, predvsem s področja elektronike in načrtovanja projektov (npr. branje datasheetov, dvomiti v zaupanje podatkom podanih na spletnih straneh, podrobno proučevanje izbranih komponent, …).

Zelo koristno je tudi, da smo se projekta lotili, kot bi to potekalo v resnici. Na začetku smo morali z ''naročniki'' priti do podpisa pogodbe in projekt izpeljati do konca. Za dosego cilja je bilo potrebno večkratno sestajanje in proučevanje možnih rešitev.

Presenetljivo je, da je veliko časa vzela zasnova projekta, ki se je izkazala za enega izmed ključnih delov projekta. Zelo pomembno je oceniti obseg projekta in delovanje zasnovati v celoti. Pri nas se je izkazalo, da je zasnovo projekta potrebno zelo podrobno proučiti, ker je v našem primeru bilo potrebno združiti več različnih tehnologij ter omogočiti delovanje ene z drugo (branje podatkov, komunikacija z več tehnologijami, …).

# 11 VIRI


[1] The Facts About Chimney Fires, CSIA, junij 2014. Dostopno na: https://www.csia.org/chimneyfires.html [22.10.2018].

[2] ESP8266EX Datashjeet, Espressif Systems, 2018. Dostopno na: https://www.espressif.com/sites/default/ files/documentation/0a-esp8266ex_datasheet_en.pdf [29.10.2018].

[3] NodeMCU documentation. MIT, 2019. Dostopno na: https://nodemcu.readthedocs.io/en/master/ [20.10.2018].

[4] STM32l4x6. ST Microelectronics, 2018 Dostopno na: https://www.st.com/en/microcontrollers/stm32l4x6.html?querycriteria=productId=LN1840 [26.11.2018] .

[5] MQ-2 Datasheet. Polulu, 2016. Dostopno na: https://www.pololu.com/file/0J309/MQ2.pdf [20.10.2018].

[6] DHT11 – Temperature and Humidity Sensor. Components 101, januar 2018. Dostopno na: https://components101.com/dht11-temperature-sensor

[7] LoRaWAN. LoRa Alliance, 2019. Dostopno na: https://lora-alliance.org/ [30.12.2019].

[8] All You Need to Know About LoRaWAN and Hot It Works. Rohit Narayan, november 2015. Dostopno na: https://www.iotleague.com/lorawan-low-power-wide-area-network/ [1.1.2019].

[9] RYLR896. Reyax Technology, 2017. Dostopno na: http://reyax.com/products/rylr896/ [11.11.2018]-

[10] Ra-01 LoRa Module. Ai-Thinker, 2017. Dostopno na: http://wiki.ai-thinker.com/_media/lora/docs/c047ps01a1_ra-01_product_specification_v1.1.pdf [11.12.2018].

[11] Wikipedia, Firebase. Dostopno na: https://en.wikipedia.org/wiki/Firebase [13.12.2018].

[12] Adafruit DHT Humidity and Temperature Sensor Library. PaintYourDragon, 2018. Dostopno na: https://github.com/adafruit/DHT-sensor-library [20.10.2018].

[13] Get Started With XL1276-D01. Shen7, 2017 Dostopno na: https://www.instructables.com/id/Get-Started-With-XL1276-D01-LoRa-433MHz-SPI-Using-/ [15.12.2018].

[14] Battery Life Calculator. Oregon Embedded, 2018. Dostopno na: http://www.oregonembedded.com/batterycalc.htm [27.12.2018].

[15] How to Use Mosfet. Oscar Liang, 2018. Dostopno na: https://oscarliang.com/how-to-use-mosfet-beginner-tutorial/ [15.1.2019].

[16] Crypto Wiki, XXTEA. Dostopna na: http://cryptography.wikia.com/wiki/XXTEA [10.1.2019].

[17] XXTEA Encryption and Decryption Library for Arudino. Boseji, november 2018. Dostopno na: https://github.com/boseji/xxtea-lib [10.1.2019].

[18] LoRa Shield Library. Pdelmo, november 2015 [10.1.2019].