**JELGAVAS TEHNIKUMS**

**Aleks Markuss Balodis**

**Datu ieguve un apmaiņa izmantojot mikro kontrolierus**

**Kvalifikācijas darbs**

**kvalifikācijas ieguvei**

**programmēšanas tehniķis**

**Darba izpildītājs:  410.gr.izgl. Aleks Markuss Balodis**

***paraksts, datums***

**Jelgava**

**2022**

ANOTĀCIJA

Balodis Aleks Markuss. Datu ieguve un apmaiņa izmantojot mikro kontrolierus: kvalifikācijas darbs. Jelgava: JT, 2022. 43 lpp., 13 att., 2 tab., 4 informācijas avoti, 4 pielikumi.

Mikro kontrolieri plaši tiek izmantoti mūsdienās dažādās ierīcēs, lai veiktu vienkāršas funkcijas, kur nav nepieciešama liela veiktspēja. Darba mērķis ir ievākt datus par laikapstākļiem ar mikro kontrolieriem un, izmantojot bezvadu komunikāciju, nosūtīt tos uz serveri, kas tos saglabās datubāzē un attēlos lietotājam viegli pārredzamā veidā mājaslapā.

Mikro kontrolieri paši nespēj izmērīt temperatūru, gaismas daudzumu vai nosūtīt informāciju bez vadiem, bet vien apstrādāt loģiku, tāpēc tiek pievienoti dažādi moduļi vai komponentes. Komponentes kā *LED* tiek pievienotas izmantojot *GPIO* kontaktus, bet moduļi kā, piemēram *nRF24L01* bezvadu modulis, izmanto komunikācijas protokolu *SPI*.

Darba veikšanai nepieciešams būs izmantot dažādas programmēšanas valodas kā *C*. Koda rediģēšanai, tiks izmantots *Visual Studio Code* un *Arduino IDE*. Versiju kontroles sistēmai tiks izmantots *GIT*.

ANNOTATION

Balodis Aleks Markuss. Data collection and exchange using microcontrollers – qualification work. Jelgava: JT, 2022, 43 pages, 13 images, 2 tables, 4 used information sources, 4 attachments.

Microcontrollers are widely used today in various devices to perform basic functions which do not require high performance. The purpose of my final year project is to harvest weather data using microcontrollers and then using wireless communication send the harvested data to a server, which would store the data in a database.

Microcontrollers by themselves cannot measure temperature, light or send information wirelessly. They can only process logic, that is why extra modules or components are added to them. Components like *LEDs* are added using *GPIO* pins, but modules like the *nRF24L01* wireless module use a communication protocol, in this case *SPI*.

For this project I will need to use a programming language like *C*. To edit the code, I will be using *Visual Studio Code* and the *Arduino IDE*. For version control I will be using *GIT*.

SATURS

[IEVADS 6](#_Toc106284845)

[1. PROGRAMMATŪRAS PRASĪBU SPECIFIKĀCIJA 7](#_Toc106284846)

[1.1 Ievads 7](#_Toc106284847)

[1.1.1 Dokumentācijas mērķis 7](#_Toc106284848)

[1.1.2 Produkta sfēra 7](#_Toc106284849)

[1.1.3 Dokumentācijas pārskats 7](#_Toc106284850)

[1.2 Vispārīgs apraksts 8](#_Toc106284851)

[1.2.1 Produkta perspektīva 8](#_Toc106284852)

[1.2.2 Produkta funkcijas 8](#_Toc106284853)

[1.2.3 Programmatūras gaita 8](#_Toc106284854)

[1.2.4 Mērķauditorija 8](#_Toc106284855)

[1.3 Funkcionālās prasības 9](#_Toc106284856)

[1.3.1 Datu ievākšana 10](#_Toc106284857)

[1.3.2 Datu pārraide 10](#_Toc106284858)

[1.3.3 Datu saņemšana un ievietošana datubāzē 11](#_Toc106284859)

[1.4 Nefunkcionālās prasības 12](#_Toc106284860)

[1.4.1 *Arduino* programmsaderība 12](#_Toc106284861)

[1.4.2 Atvērtā pirmkoda programmatūra 12](#_Toc106284862)

[2. IZSTRĀDEI IZMANTOTĀS TEHNOLOĢIJAS UN PROGRAMMATŪRA 13](#_Toc106284863)

[2.1 *Arduino Uno* 13](#_Toc106284864)

[2.2 *nRF24L01* bezvadu modulis 13](#_Toc106284865)

[2.3 *DHT-22* mitruma un temperatūras sensori 13](#_Toc106284866)

[2.4 Fotopretestība 13](#_Toc106284867)

[2.5 *C / C++* 13](#_Toc106284868)

[2.6 *Visual Studio Code* 14](#_Toc106284869)

[*2.6.1* *PlatformIO* 14](#_Toc106284870)

[*2.6.2* *C / C++ Syntax* 15](#_Toc106284871)

[2.7 *GitHub* 15](#_Toc106284872)

[3. PRODUKTA KOMPONENŠU APRAKSTS 16](#_Toc106284873)

[3.1 Mikro kontroliera pirmkoda komponentes 16](#_Toc106284874)

[3.1.1 Bezvadu moduļa nRF24L01 bibliotēkas funkcijas 16](#_Toc106284875)

[3.1.2 Klienta mikro kontroliera funkcijas 19](#_Toc106284876)

[3.1.3 Centrālā mikro kontroliera funkcijas 19](#_Toc106284877)

[3.2 *Python* pirmkoda funkcijas 20](#_Toc106284878)

[4. LIETOTĀJA ROKASGARĀMATA 21](#_Toc106284879)

[5. DATUBĀZES TABULA DATU GLABĀŠANAI 27](#_Toc106284880)

[SECINĀJUMI 28](#_Toc106284881)

[Izmantoto informācijas avotu saraksts 29](#_Toc106284882)

[PIELIKUMI 30](#_Toc106284883)

akronīmi un saīsinājumI

|  |  |
| --- | --- |
| *WEB* | Tehnoloģijas un dažādi rīki, kas tiek izmantoti komunikācijas procesā starp dažādām ierīcēm tīmeklī. |
| *C / C++* | Programmēšanas valoda, kas izmantota pārsvarā sistēmu un *embedded* programmēšanā. Tā ir efektīva resursu utilizācijā un elastīga. |
| *nRF24L01* | Bezvadu modulis ar ko iespējams saņemt vai sūtīt datus. 2.4GHz frekvencē. |
| *DHT-22* | Temperatūras un mitruma sensors, ko iespējams pielikt mikro kontrolierim. |
| *BIOS* | Datora programmatūra, kas inicializē aparatūru un nodod tās kontroli operētājsistēmai. |
| *GitHub* | Microsoft veidota platforma versiju kontrolei un pirmkoda menedžmentam, kas izmanto *GIT.* |
| *Visual Studio Code / VSC* | Microsoft veidota teksta rediģēšanas programma, kam ir iespējams pievienot dažādus paplašinājumus. |
| *Arduino* | Atvērta pirmkoda platforma, mikro kontrolierim. Firma, kas izstrādā mikro kontrolierus, lai veidotu dažādas ierīces. |
| *Mongo DB* | Datubāzes programma. |
| *MCU* | Saīsinājums vārdam mikro kontrolieris. |
| *SPI* | Komunikācijas specifikācija, kas parasti tiek izmantota *embedded* sistēmās. |
| *Embedded* | Sistēma, kas ir paredzēta specifiskai funkcijai kam ir integrēta atmiņa, procesors un izejas/ieejas. |
| *Serial RS-232* | Interfeiss, kas ļauj pārsūtīt datus no mikro kontroliera uz citu ierīci. |
| *Python* | Augsta līmeņa, interpretēta programmēšanas valoda. |
| *Chip Select* | Signāls, ko izmanto *SPI* interfeiss, lai atlasītu ierīci. |
| *Chip Select Not* | Signāls, ko izmanto *SPI* interfeiss, lai iespējotu ierīci. |
| *WinRAR* | Failu arhivēšanas programma. |

# **IEVADS**

**Darba tēma** “Datu ieguve un apmaiņa izmantojot mikro kontrolierus” tika izvēlēta, dēļ klienta nepieciešamības ievākt datus par temperatūru, mitrumu un apgaismojumu un ar bezvadu savienojumu pārsūtīt šos datus uz datubāzi.

Šādu datu ievākšana ļauj veikt automatizāciju dažādās veidos, piemēram, mājas apkures regulācija izmantojot ievāktos datus vai siltumnīcas apstākļu novērošanu, lai uzlabotu augu apstākļus.

**Darba mērķis:** Izveidot sistēmu, kas ievāc datus par apkārtnes temperatūru, mitrumu un apgaismojumu un šos ievāktos datus ievieto datubāzē.

**Darba uzdevumi:**

* Izveidot ierīci, kas ievāks datus par apkārtni – temperatūru, mitrumu un apgaismojumu.
* Ierīce, kas ievāc datus ar bezvadu savienojumu tos, pārraida uz centrālu ierīci.
* Ierīce, kas datus saņem, ievieto tos datubāzē ar datora starpniecību.
* Pārbaude, vai saņemtie dati ir neskarti.

# **PROGRAMMATŪRAS PRASĪBU SPECIFIKĀCIJA**

## Ievads

### Dokumentācijas mērķis

Programmatūras prasību specifikācijas nolūks ir aprakstīt prasības uzdevumu plānošanas informācijas sistēmā.

Dokuments paredzēts izstrādātājiem un programmatūras lietotājiem, lai varētu izprast programmatūras funkcijas, darbību un specifiku. Dokumentācija palīdzēs tās lietotājiem integrēt programmatūru veiksmīgi savā projektā.

### Produkta sfēra

Programmas produkta nosaukums ir “*MCU Weather*”, ko var izmantot, lai ievāktu datus par temperatūru, mitrumu un apgaismojumu, un nosūtīt tos izmantojot bezvadu savienojumu.

Programmu var pielietot dažādos veidos, dažas aplikācijas ir mājas apkures regulācija izmantojot ievāktos datus vai siltumnīcas apstākļu novērošanu, lai uzlabotu augu apstākļus.

### Dokumentācijas pārskats

Šī dokumentācija sevī ietver *“MCU Weather”* specifikāciju, izstrādē izmantotās tehnoloģijas, produkta komponenšu aprakstu, lietotāja rokasgrāmatu un vispārējo aprakstu.

Pirmā nodaļa ietver sevī:

* Ievadu, kurā aprakstīts programmatūras nolūks, darbības sfēra un definīcijas.
* Vispārējo aprakstu, kas sevī ietver produkta perspektīvu, produkta funkcijas, programmatūras gaitu un mērķauditoriju.
* Funkcionālās prasības, kas apraksta programmatūras funkcijas, kā tās lietot un to darbību.
* Nefunkcionālās prasības, kas apraksta aparatūras saskarsmi.

Otrā nodaļa – Produkta izstrādē izmantoto tehnoloģiju apraksts un pielietojums.

Trešā nodaļa – Produkta pirmkoda funkciju apraksts un pielietojums.

Ceturtā nodaļa – Lietotāja rokasgrāmata, kas detalizēti izskaidro kā lietot programmatūru.

## Vispārīgs apraksts

### Produkta perspektīva

Izstrādātai programmai nākotnē ir iespējams pievienot datu vizualizāciju, kas ļaus lietotājam iegūtos datus vieglāk uztvert.

Papildus nākotnē arī iriespējams pievienot integrāciju ar mākoni, kur var apskatīt, piemēram, apstākļus savās mājās attālināti, vai izmantojot šos datus, izveidot automatizāciju, kas regulē, mājas apstākļus ar apkures sistēmas palīdzību.

### Produkta funkcijas

Izstrādātai programmatūrai ir vairākas funkcijas:

1. Veikt temperatūras un gaisa mitruma mērījumus.
2. Veikt apgaismojuma mērījumus.
3. Nosūtīt ievāktos datus ar bezvadu savienojumu
4. Ceturtā funkcija ir nosūtīt datus uz citu mikro kontrolieri kurš tos apstrādā un pārraida datoram/serverim.

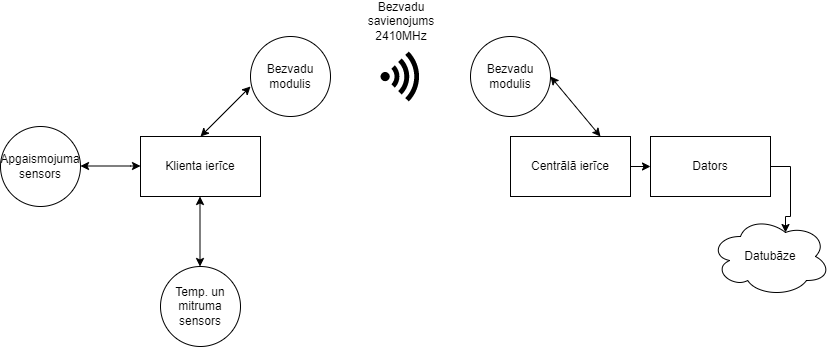
### Programmatūras gaita

Programmai uzsākot darbu, ierīce ievāks datus par temperatūru, mitrumu un apgaismojumu. Šie dati ar bezvadu savienojumu tiks pārsūtīti uz centrālo ierīci, kas ar datora starpniecību datus ievietos datubāzē. Datu pārsūtīšana notiks vienu reizi sekundē, un centrālā ierīce būs gatava šos datus saņemt jebkad.

### Mērķauditorija

Programmatūra ir paredzēta cilvēkiem, kam ir nepieciešamība ievākt datus par temperatūru, mitrumu un apgaismojumu. Šos datus var izmantot, lai veiktu automatizāciju mājās, siltumnīcās un citviet.

## Funkcionālās prasības

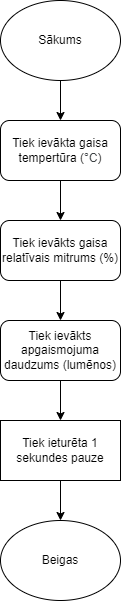


1. att. **Sistēmas modelis**

Programmatūras funkcionālās prasības ietilpst šāda funkcionalitāte un sistēmas darbības principi:

* Ierīce ievāc sekojošos datus par savu apkārtni: temperatūru celsijā (°C), gaisa relatīvo mitrumu procentos un apgaismojumu lūmenos.
* Ierīce, kas ievāc datus tos pārraida ar bezvadu savienojuma palīdzību 2410MHz frekvencē.
* Dati tiek sūtīt periodiski, kur periods ir 1 sekunde.
* Ierīce, kas saņem datus, ir vienmēr gatava to darīt.
* Ierīce, kas saņem datus, ievieto šos datus datubāzē ar datora starpniecību.

### Datu ievākšana



2. att. **Datu ievākšanas diagramma**

Ierīcei ir jāspēj ievākt sekojošos datus par savu apkārtni:

* Gaisa temperatūra (celsijā)
* Gaisa relatīvais mitrums procentos
* Apkārtnes apgaismojuma līmeni (lūmenos)

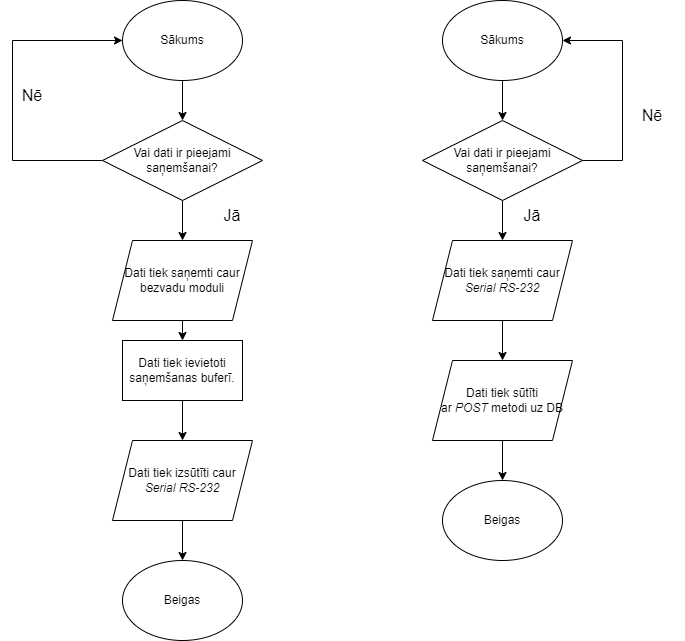
Temperatūra ir jāspēj noteikt robežās no -40°C līdz +80°C ar pieļaujamo kļūdu ±0.5°C. Gaisa relatīvo mitrumu ir jāspēj noteikt robežās no 0% līdz 100% ar pieļaujamo kļūdu ±1%. Dati tiek ievākti periodiski ar periodu 1 sekunde.

### Datu pārraide

Ierīcei, kas ievāc datus nosūta tos periodiski, kur periods ir 1 sekunde. Datiem ir jātiek pārraidītiem ar bezvadu savienojumu, kas darbojas 2410MHz frekvencē. Ierīcei, kas nodrošina bezvadu savienojumu datu pārraidei ir jābūt spējīgai pārraidīt datus atvērtā laukā līdz 100 metriem.

Dati, kas tiks pārraidīti: temperatūra (celsijā), gaisa relatīvais mitrums (procentos) un apkārtnes apgaismojuma līmenis (lūmenos)

### Datu saņemšana un ievietošana datubāzē



3. att. **Datu saņemšanas un ievietošanas DB diagramma**

Ierīcei, kas saņem datus ir jābūt gatavai to darīt jebkad. Pēc datu saņemšanas dati tiek pārraidīti caur *Serial RS-232* interfeisu uz datoru. Uz datora darbojas *Python* programma, kas datus ievieto datubāzē. Datubāzē ir jātiek ievietotiem šādiem datiem: temperatūra (celsijā), gaisa relatīvais mitrums (procentos), apkārtnes apgaismojuma līmenis (lūmenos) un datumam ar laiku (formāts: GGGG-MM-DDTHH:MM:SS:MSZ)

## Nefunkcionālās prasības

### *Arduino* programmsaderība

Izstrādājot programmatūru, tai ir jābūt saderīgai ar *Arduino* platformu. Tas nodrošina ērtu lietošanu un integrāciju ar citiem projektiem.

Nākotne ir arī iespēja izmantot *Arduino Cloud*, kas ļaus lietotājam piekļūt ievāktajiem datiem no jeb kurienes un veikt automatizāciju.

### Atvērtā pirmkoda programmatūra

Izstrādātai programmatūrai ir jābūt ar atvērtu pirmkodu. Programmatūras pirmkodam ir jābūt pieejamam *GitHub*.

Atvērts pirmkods ļauj citiem lietot, rediģēt un apskatīt programmas pirmkodu. Ieguvums ir, ka citi cilvēki var apskatīt pirmkodu un veikt uzlabojumus vai izlabot pamanītu kļūdu, ar oriģinālā autora atļauju.

# **IZSTRĀDEI IZMANTOTĀS TEHNOLOĢIJAS UN PROGRAMMATŪRA**

## *Arduino Uno*

*Arduino Uno* ir atvērta pirmkoda mikro kontroliera plate, kas ir bāzēta uz *Microchip ATmega328P* mikro kontrolieri. *Arduino Uno* tika izvēlēts, jo tas izpilda divas nefunkcionālās prasības – saderība ar *Arduino* platformu un atvērta pirmkoda programmatūra.

Papildus tam šim mikro kontrolierim var pievienot dažādus moduļus, kas nodrošinās temperatūras, mitruma un apgaismojuma mērīšanu, kā arī bezvadu savienojumu.

## *nRF24L01* bezvadu modulis

*nRF24L01* bezvadu modulis tiks pieslēgts *Arduino Uno* mikro kontrolierim izmantojot *SPI* interfeisu. Šis modulis nodrošinās bezvadu datu saņemšanu un sūtīšanu.

Vadoties pēc funkcionālajām prasībām, šis modulis arī atbalsta datu pārraidi 2410 MHz frekvencē un spēj paraidīt datus 100 metru attālumā atvērtā laukā.

## *DHT-22* mitruma un temperatūras sensori

*DHT-22* mitruma un temperatūras sensori tika izvēlēts tādēļ, ka tas ir viegli lietojams un atbilst funkcionālām prasībām. Šis sensors ir spējīgs noteikt temperatūru robežās no -40°C līdz +80°C ar pieļaujamo kļūdu ±0.5°C un gaisa relatīvo mitrumu robežās no 0% līdz 100% ar pieļaujamo kļūdu ±1%.

## Fotopretestība

Lai noteiktu apgaismojuma daudzumu lūmenos tiks, izmantota vienkārša foto pretestība, kas tiks pieslēgta *Arduino Uno*. *Arduino Uno* izmērīs šīs foto pretestības analogo vērtību un izmantojot speciālu algoritmu pārveidos to uz lūmeniem.

## *C / C++*

*C / C++* ir programmēšanas valoda, kas parasti tiek izmantota *embedded* sistēmu programmēšanā. Šī programmēšanas valoda ļauj manipulēt atmiņu zemā līmenī un tiek kompilēta specifiskai platformai. Tā rezultātā *C / C++* ir ļoti efektīva un programmas, kas tajā tiek rakstītas, ir ļoti optimizētas.

*C / C++* projektam tiek izmantota, kā galvenā programmēšanas valoda, lai veidotu pirmkodu Arduino Uno.

## *Visual Studio Code*

*Visual Studio Code (VSC)* ir Microsoft veidota teksta rediģēšanas programma, kurai ir iespējams uzstādīt dažādus paplašinājumus, kas nodrošina lietotājam nepieciešamo funkcionalitāti. *VSC* pamatā nodrošina šīs un citas funkcijas:

* integrētu komandas rindu;
* sintakses pārbaudītāju;
* *IntelliSense*, kas ļauj automātiski pabeigt rakstīto, dod informāciju par funkciju, tās parametriem un tipu un daudz ko citu.

### *PlatformIO*

*PlatformIO* ir paplašinājums priekš *Visual Studio Code*, kas ļauj izstrādāt programmatūru priekš dažādām *embedded* platformām. Šī darba ietvaros šis paplašinājums tiek lietots, lai izstrādātu programmatūru *Arduino* platformai.

*PlatformIO* nodrošina šādu funkcionalitāti:

* Koda kompilēšanai izvēlētajai platformai
* Koda atkļūdošana
* Bibliotēku viegla pievienošana
* Projektu menedžeri

### *C / C++ Syntax*

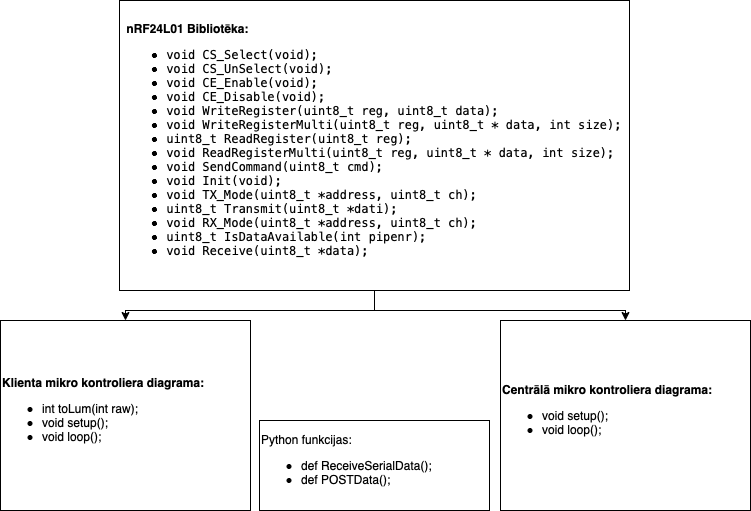
*C / C++ Syntax* paplašinājums pievieno *Visual Studio Code C / C++* programmēšanas valodas atbalstu, *IntelliSense* un atkļūdošanu.

## *GitHub*

*Github* ir Microsoft veidots hostings programmatūras izstrādei un versiju kontrolei. *GitHub* izmanto *GIT* versiju kontroli un pirmkoda pārvaldību. *GitHub* piedāvā arī vairākus sadarbības līdzekļus, piemēram, kļūdu izsekošanu, uzdevumu pārvaldību un dokumentāciju katram projektam.

Projektam tiek izmantots *GitHub*, lai nodrošinātu versiju kontroli un nodrošinātu atvērta pirmkoda prasību.

# **PRODUKTA KOMPONENŠU APRAKSTS**



3. att. **Funkciju diagramma**

## Mikro kontroliera pirmkoda komponentes

### Bezvadu moduļa nRF24L01 bibliotēkas funkcijas

**void CS\_Select(void);**

Funkcija iestata *Chip Select* signālu zemajā līmenī *SPI* interfeisā starp *MCU* un *nRF24L01*. Šī funkcija neatgriež neko un nepieprasa parametrus.

**void CS\_UnSelect(void);**

Funkcija iestata *Chip Select* signālu augstajā līmenī *SPI* interfeisā starp *MCU* un *nRF24L01*. Šī funkcija neatgriež neko un nepieprasa parametrus.

**void CE\_Enable(void);**

Funkcija iestata *Chip Select Not (Chip Enable)* signālu augstajā līmenī *SPI* interfeisā starp *MCU* un *nRF24L01*. Šī funkcija neatgriež neko un nepieprasa parametrus.

**void CE\_Disable(void);**

Funkcija iestata *Chip Select Not (Chip Enable)* signālu zemajā līmenī *SPI* interfeisā starp *MCU* un *nRF24L01*. Šī funkcija neatgriež neko un nepieprasa parametrus.

**void WriteRegister(uint8\_t reg, uint8\_t data);**

Funkcija ļauj rakstīt *nRF24L01* reģistros caur *SPI* interfeisu. Kā parametrus funkcija pieņem reģistra adresi un datus, kas tiks ierakstīti konkrētajā reģistrā. Funkcija neatgriež vērtību.

**void WriteRegisterMulti(uint8\_t reg, uint8\_t \* data, int size);**

Funkcija ļauj rakstīt konkrētā reģistrā vairākas vērtības. Funkcija neatgriež neko, bet kā parametrus pieņem reģistra adresi, rādītāju uz vērtību masīvu un masīva lielumu.

**uint8\_t ReadRegister(uint8\_t reg);**

Funkcija ļauj nolasīt *nRF24L01* reģistra vērtību caur *SPI* interfeisu. Kā parametru funkcija pieņem nolasāmā reģistra adresi. Kā vērtību funkcija atgriež veselu skaitli, kas atbilst reģistra vērtībai.

**void ReadRegisterMulti(uint8\_t reg, uint8\_t \* data, int size);**

Funkcija ļauj nolasīt vairākas vērtības no konkrēta reģistra caur *SPI* interfeisu. Funkcija neatgriež neko, bet kā parametrus pieņem nolasāmo reģistru, rādītāju uz masīvu, kur ievietot nolasītās vērtības un masīva lielumu.

**void SendCommand(uint8\_t cmd);**

Funkcija caur *SPI* interfeisu ļauj nosūtīt komandu uz *nRF24L01*. Funkcija vērtību neatgriež, bet kā parametru pieprasa nosūtāmo komandu.

**void Init(void);**

Funkcija sagatavo *nRF24L01* bezvadu moduli darbībai. Vispirms konfigurācija tiek nodzēsta iestatot visas vērtības uz 0. Pēc tam tiek izslēgta *Auto-Acknowledgment* funkcija, izslēgtas visas datu saņemšanas līnijas, saņemšanas/sūtīšanas līniju platums tiek iestatīts uz 5 baitiem, *Auto-Retransmit* tiek iestatīts uz 250 mikrosekundēm, moduļa frekvence tiek iestatīta uz 2400 MHz un ātrums uz 2Mbps. Funkcija neatgriež vērtību un nepieprasa parametrus.

**void TX\_Mode(uint8\_t \*address, uint8\_t ch);**

Funkcija iestata *nRF24L01* bezvadu moduli sūtītāja režīmā. Kā parametrus funkcija pieprasa rādītāju uz sūtīšanas adreses masīvu un moduļa frekvenci. (Frekvence = 2400 + ch).

**uint8\_t Transmit(uint8\_t \*dati);**

Funkcija pārraida datus, kas atrodas datu sūtīšanas buferī. Atgriež 1, ja dati ir veiksmīgi nosūtīti – citādi 0. Kā parametru funkcija pieņem rādītāju uz sūtāmo datu masīvu. Šim masīva lielumam ir jābūt 32.

**void RX\_Mode(uint8\_t \*address, uint8\_t ch);**

Funkcija iestata *nRF24L01* bezvadu moduli saņēmēja režīmā. Kā parametrus funkcija pieprasa rādītāju uz saņemšanas adreses masīvu un moduļa frekvenci. (Frekvence = 2400 + ch).

**uint8\_t IsDataAvailable(int pipenr);**

Funkcija pārbauda vai ir dati konkrētajā reģistrā. Ja ir dati konkrētā reģistrā, funkcija atgriež 1, citādi 0. Kā parametru pieņem saņemšanas reģistra numuru.

**void Receive(uint8\_t \*data);**

Funkcija ļauj saņemt datus. Neatgriež vērtību. Kā parametru pieņem rādītāju uz masīvu, kur glabāt saņemtos datus. Šim masīva garumam ir jābūt 32.

### Klienta mikro kontroliera funkcijas

**int toLum(int raw);**

Funkcija pārveido foto pretestības analogo vērtību uz apgaismojuma daudzumu lūmenos. Šī funkcija pieprasa kā parametru veselu skaitli - analogo vērtību un atgriež veselu skaitli – apgaismojuma daudzumu lūmenos.

**void setup();**

Funkcija izpildās vienu reizi, kad mikro kontrolieris ieslēdzas. Tiek ieslēgts *Serial* interfeiss ar ātrumu 9600, tad inicializē *DHT-22* sensoru, *SPI* interfeisu un *nRF24L01* bezvadu moduli, tad to iestata sūtīšanas režīmā.

**void loop();**

Funkcija izpildās bezgalīgā ciklā kamēr mikro kontrolieris ir ieslēgts. Tiek ievākti dati par apgaismojumu, temperatūru un mitrumu. Šie dati tiek ievietoti sūtīšanas buferī jeb masīvā un tie tiek pārraidīti uz centrālo mikro kontrolieri.

### Centrālā mikro kontroliera funkcijas

**void setup();**

Funkcija izpildās vienu reizi, kad mikro kontrolieris ieslēdzas. Tiek ieslēgts *Serial* interfeiss ar ātrumu 9600, gad inicializē *nRF24L01* bezvadu moduli, tad to iestata saņemšanas režīmā.

**void loop();**

Funkcija izpildās bezgalīgā ciklā kamēr mikro kontrolieris ir ieslēgts. Tiek pārbaudīts, vai dati ir pieejami saņemšanai, gadījumā ja ir pieejami dati saņemšanai, tad tos ielasa saņemšanas buferī, un uzrāda caur *Serial* interfeisu.

## *Python* pirmkoda funkcijas

**def ReceiveSerialData();**

Funkcija ievāc datus caur *Serial RS-232* savienojumu un ievieto tos mainīgos, lai varētu izmantot citviet programmā. Funkcija neatgriež datus un nepieprasa parametrus.

**def POSTData();**

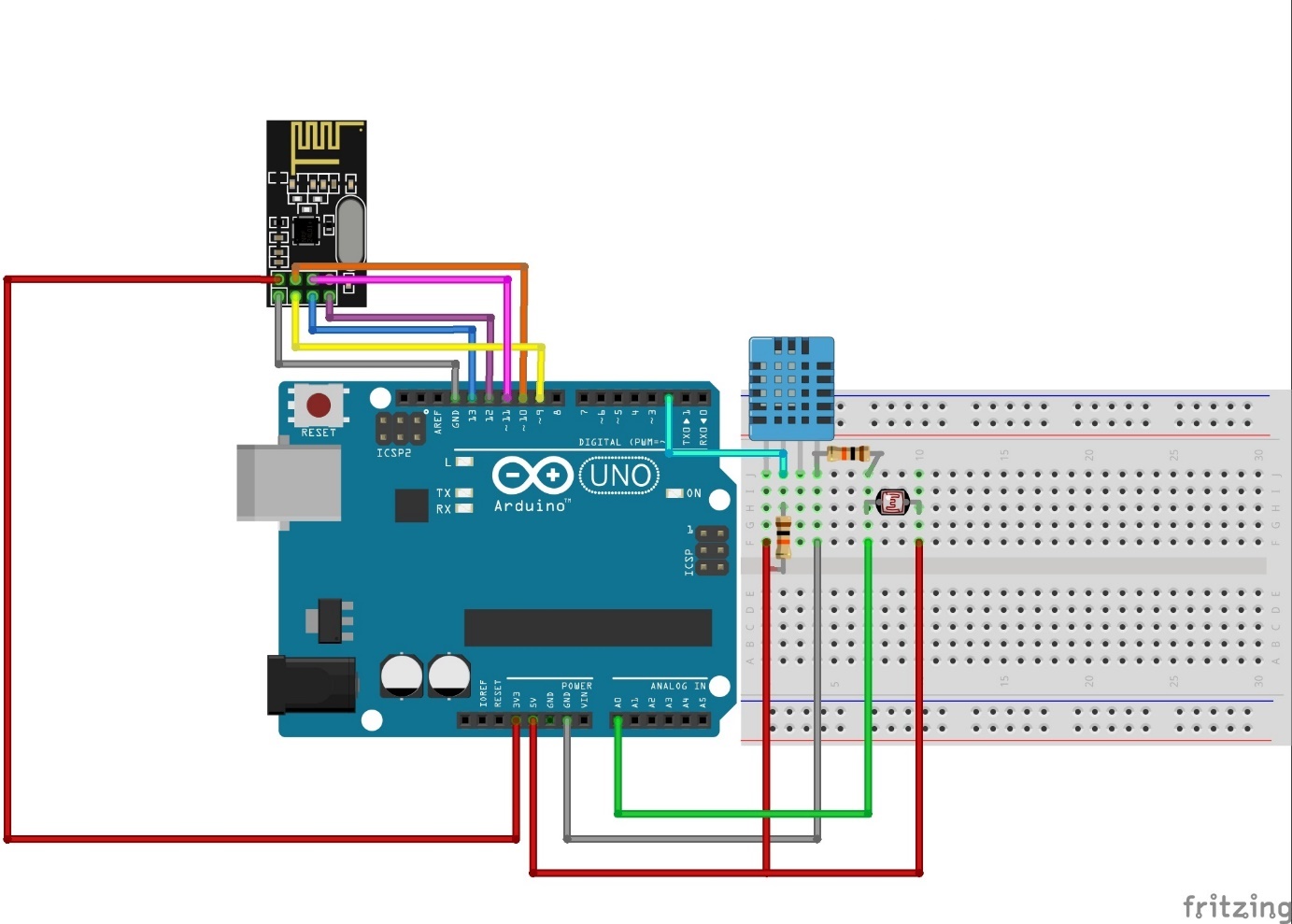
Šī funkcija ievāktos datus pārraida, uz datubāzi izmantojot post metodi. Funkcija neatgriež datus un nepieprasa parametrus.

# **LIETOTĀJA ROKASGARĀMATA**

Lietotāja rokasgrāmata ir domātā kā instruktāža lietotājam, kur tiks izstāstīts kā saslēgt, sakonfigurēt un izmantot programmu *“MCU Weather”*. Pirms sākat pārliecināties, ka jums ir sekojošās fiziskās komponentes:

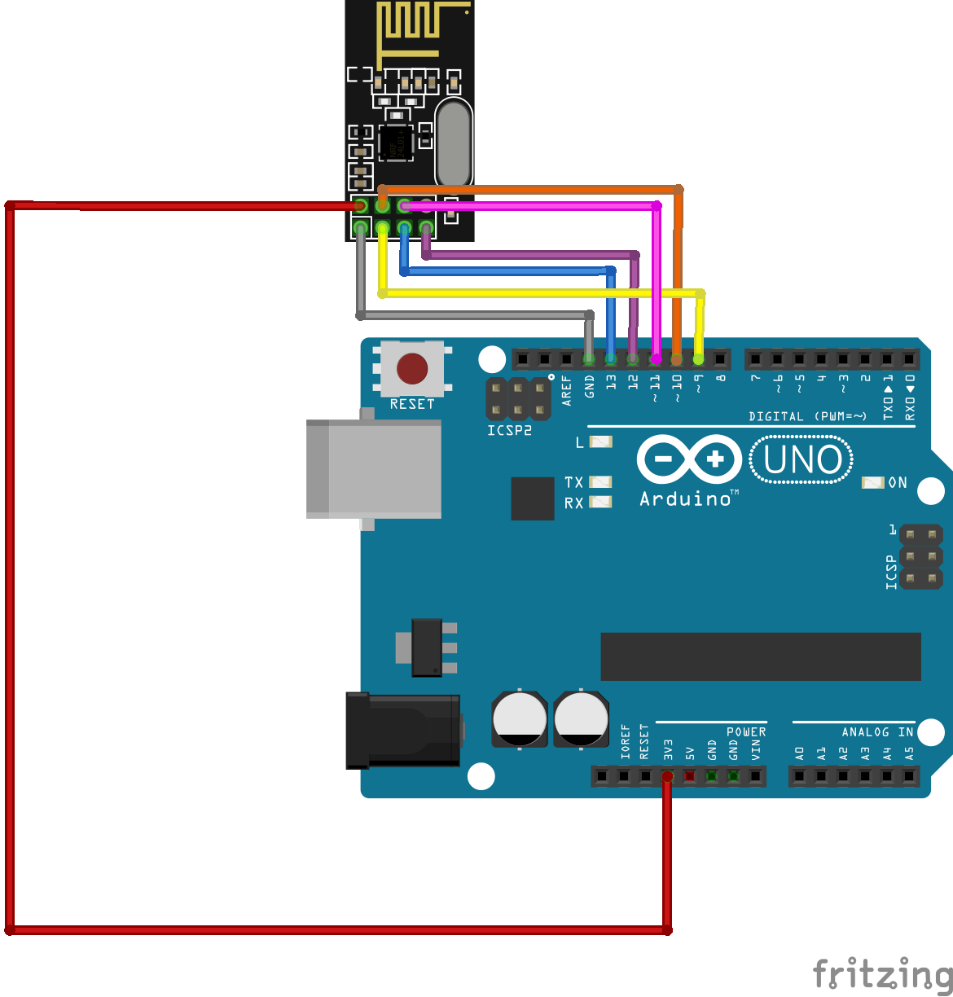
* *Arduino Uno* mikro kontroliera plate 2 gab.
* *nRF24L01* bezvadu modulis 2 gab.
* *DHT-22* temperatūras un mitruma sensors 1 gab.
* Foto pretestība 1 gab.
* 10KOhm pretestības 1 gab.
* Maketēšanas plate 1 gab.
* Maketēšanas plates vadi

1. Saslēdziet klienta mikro kontrolieri vadoties pēc shēmas, kas parādīta 4. attēlā.



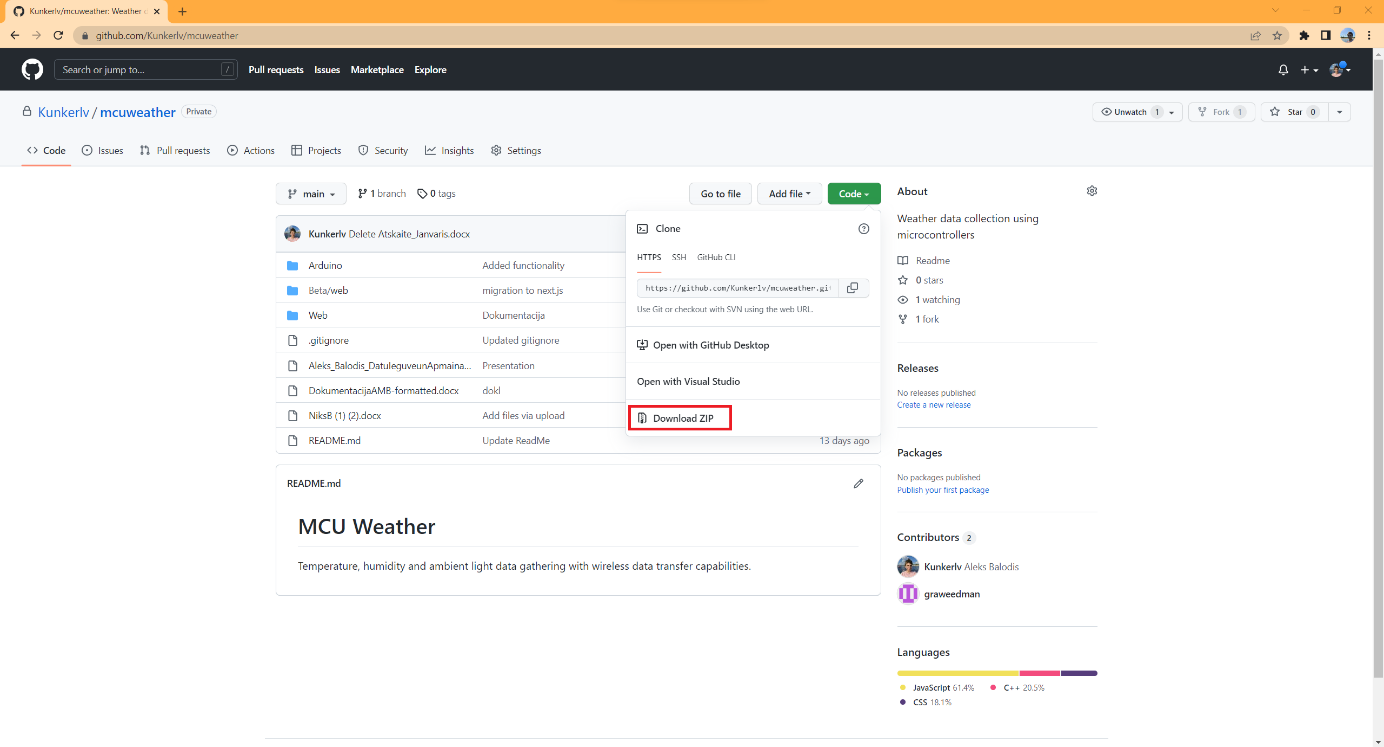
*4. att.*

1. Saslēdziet centrālo mikro kontrolieri vadoties pēc shēmas, kas parādīta 5. attēlā.



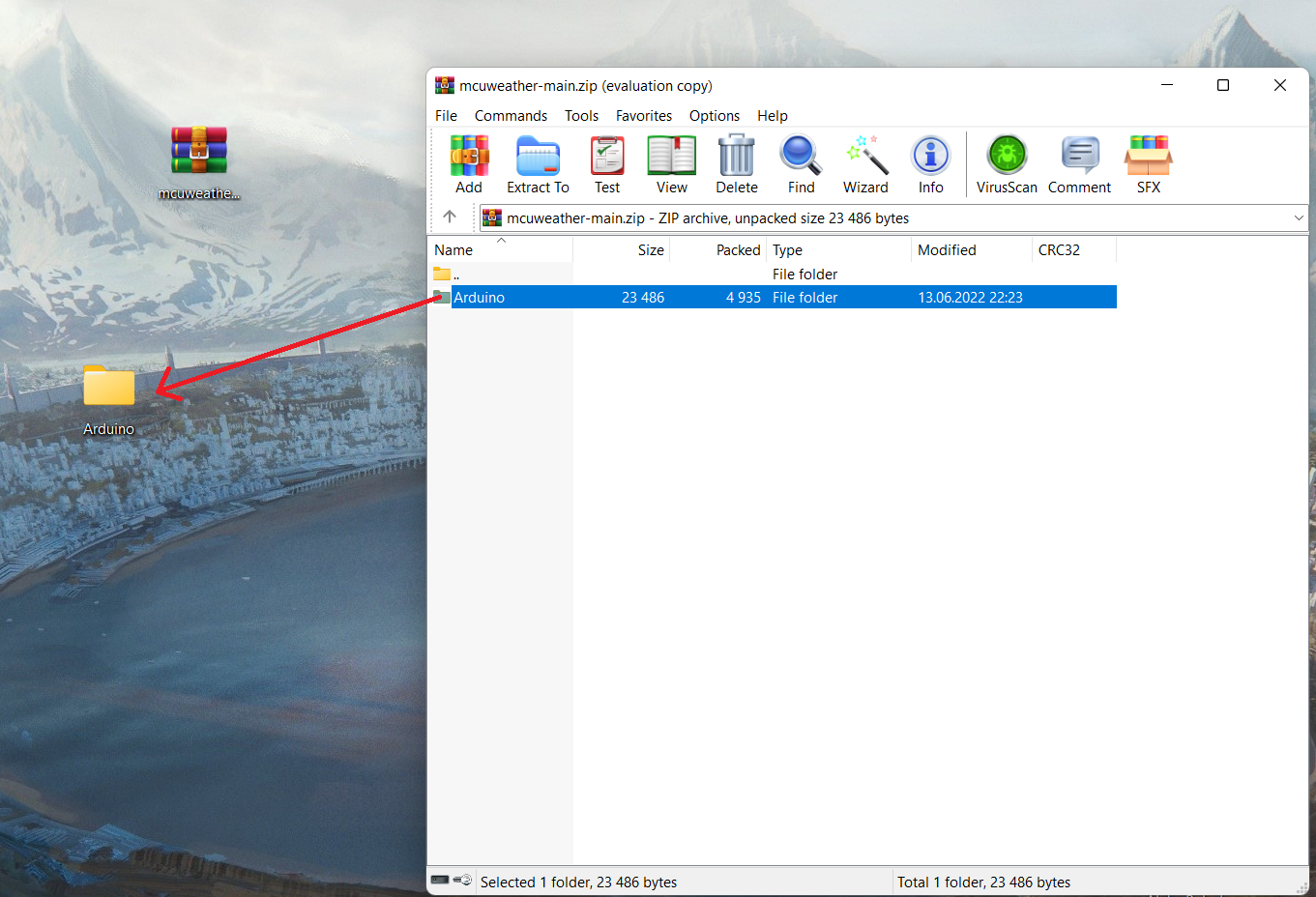
*5. att.*

1. Pārliecināties, ka jums uz datora ir ieinstalēts *Visual Studio Code* ar paplašinājumu *PlatformIO*. Ja tas nav izdarīts veiciet instalāciju sekojot *Visual Studio Code* un *PlatformIO* dokumentācijai.
2. Atveriet interneta pārlūku un dodaties uz hipersaiti https://github.com/Kunkerlv/mcuweather , spiežat zaļo pogu *“Code”*  un, tad spiežat uz *“Download ZIP”.* Skatīt 6. attēlu



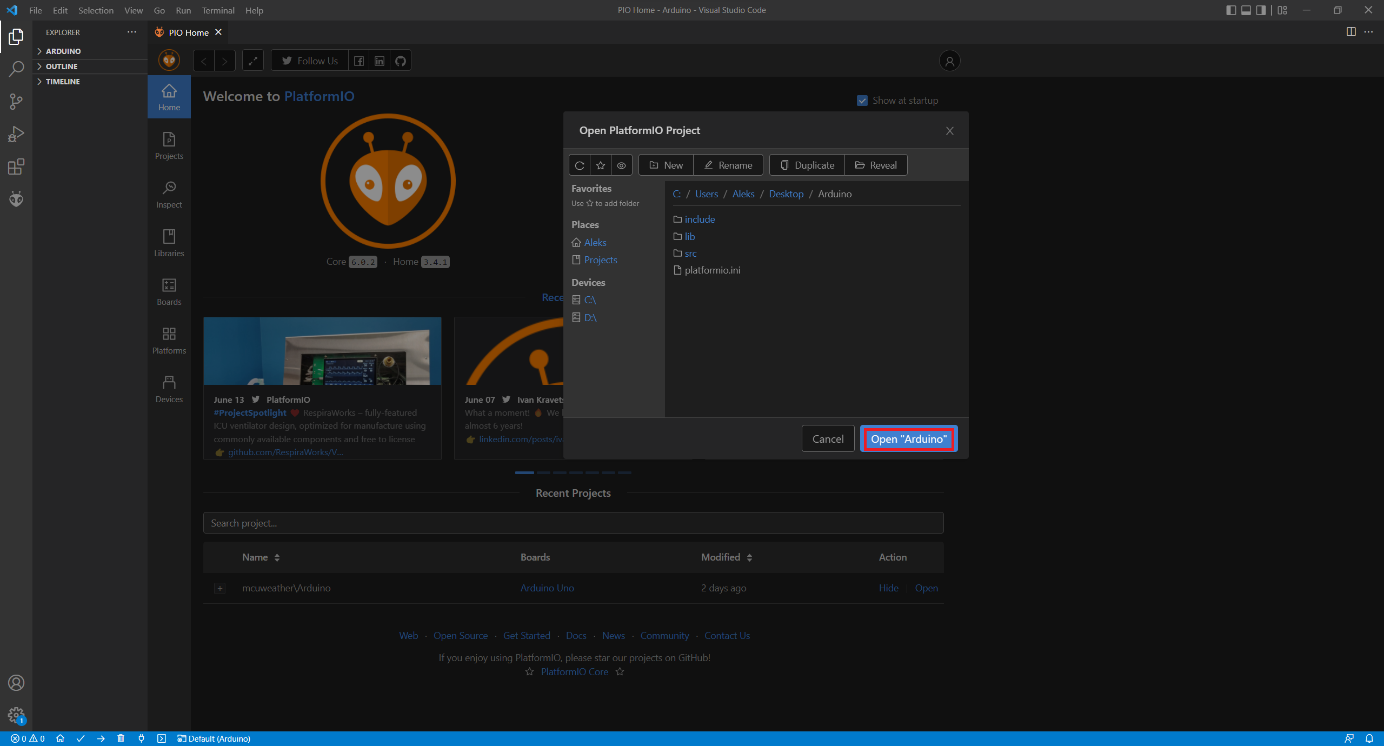
*6.att.*

1. Izmantojot failu arhivēšana programmu, piemēram, “*WinRAR*”, izvelciet lejupielādētā arhīva saturu uz darbvirsmu. Skatīt 7. attēlu



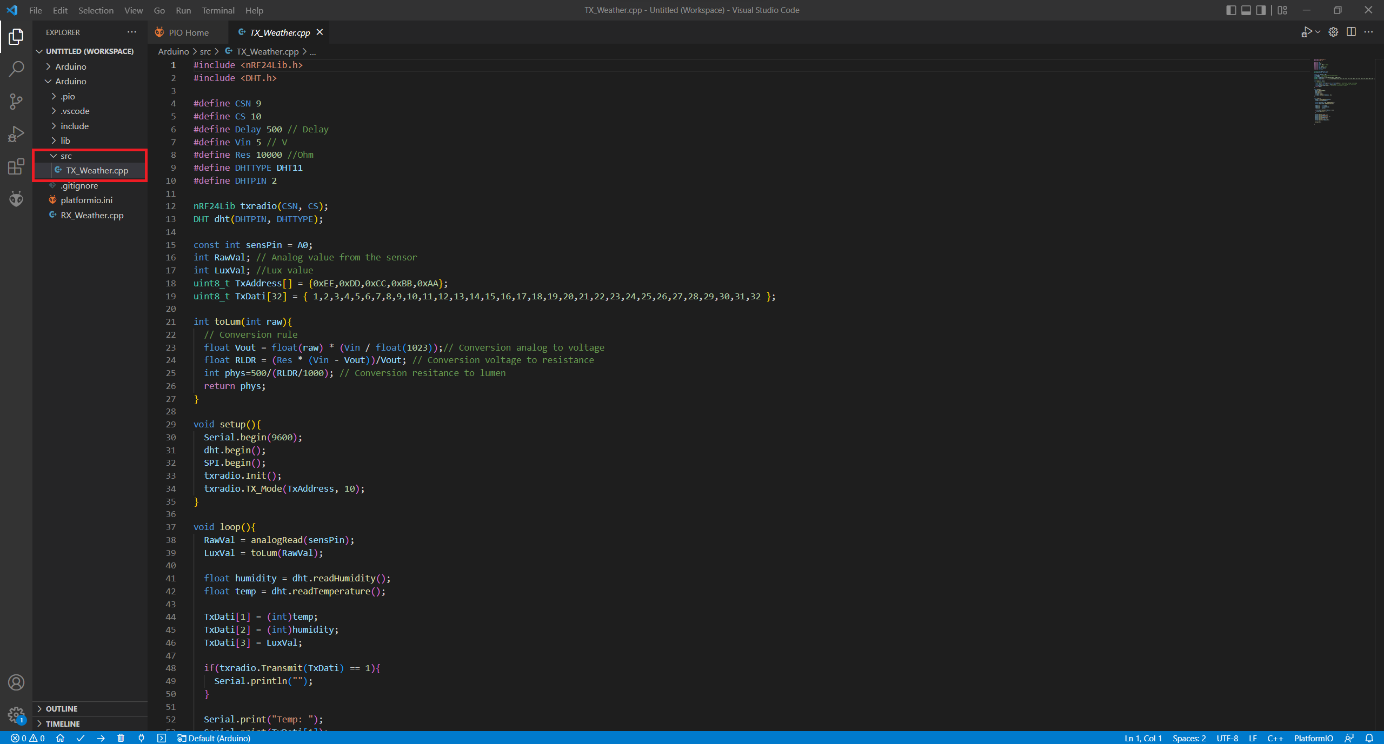
*7. att.*

1. Atveriet *Visual Studio Code*, un *PlatformIO* panelī spiežiet *“Open Project”* un izvēlaties iepriekš izvilkto mapi. Skatīt 8. attēlu.



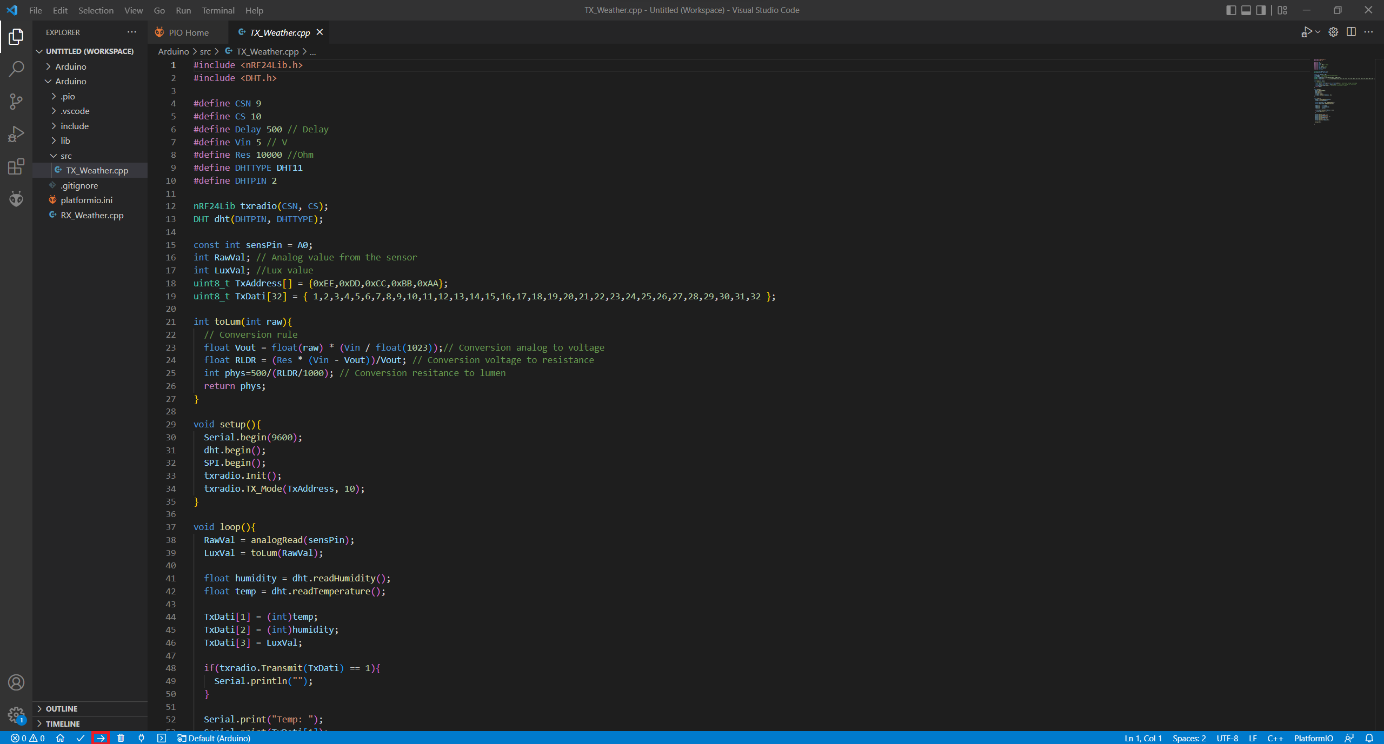
*8. att.*

1. Pievienojiet ar USB kabeli datoram klienta mikro kontrolieru kuru saslēdzāt 1. solī.
2. Iekš *Visual Studio Code* pārliecināties, ka *“src”* mapīte ir tikai fails TX\_Weather.cpp. Skatīt 9. attēlu.



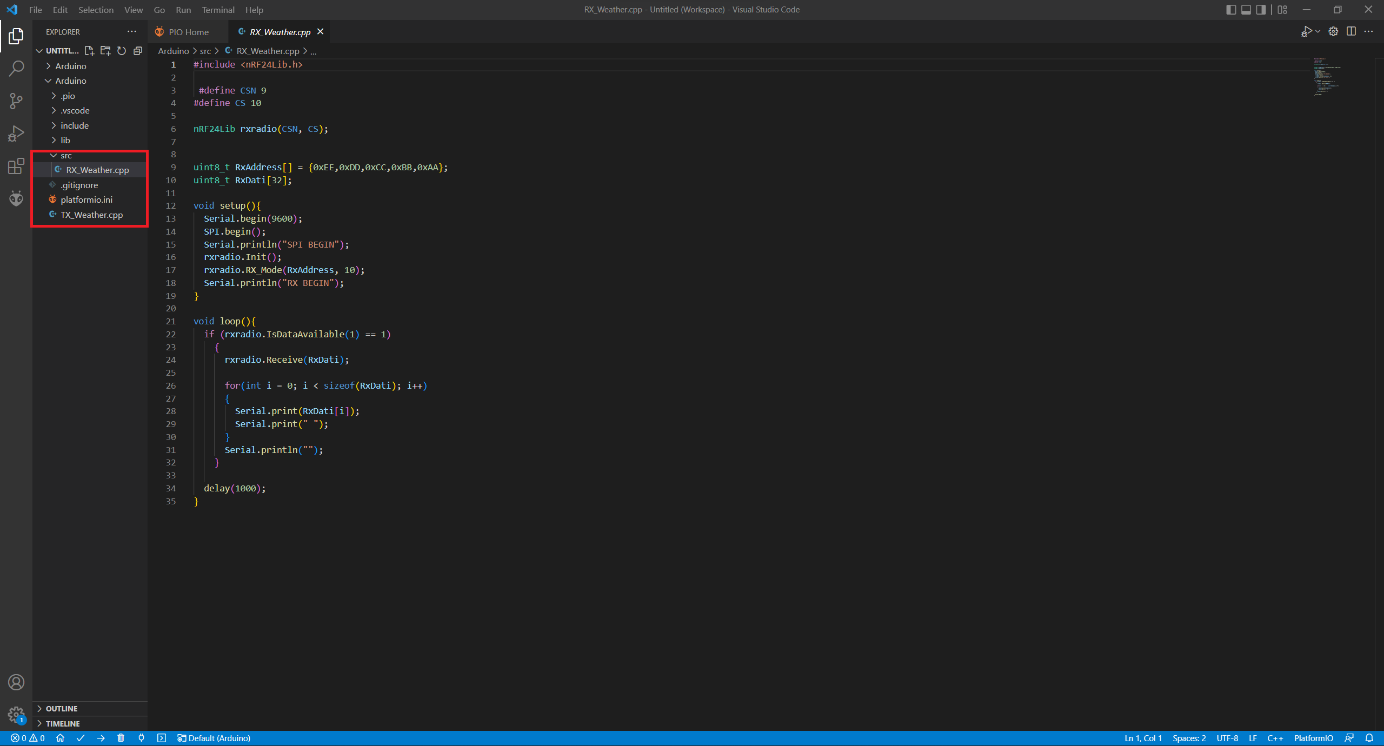
*9. att.*

1. Iekš *Visual Studio Code* spiediet pogu *“Upload”,* kas atrodas ekrāna apakšā un gaidiet kamēr process būs pabeigts. Skatīt 10. attēlu.



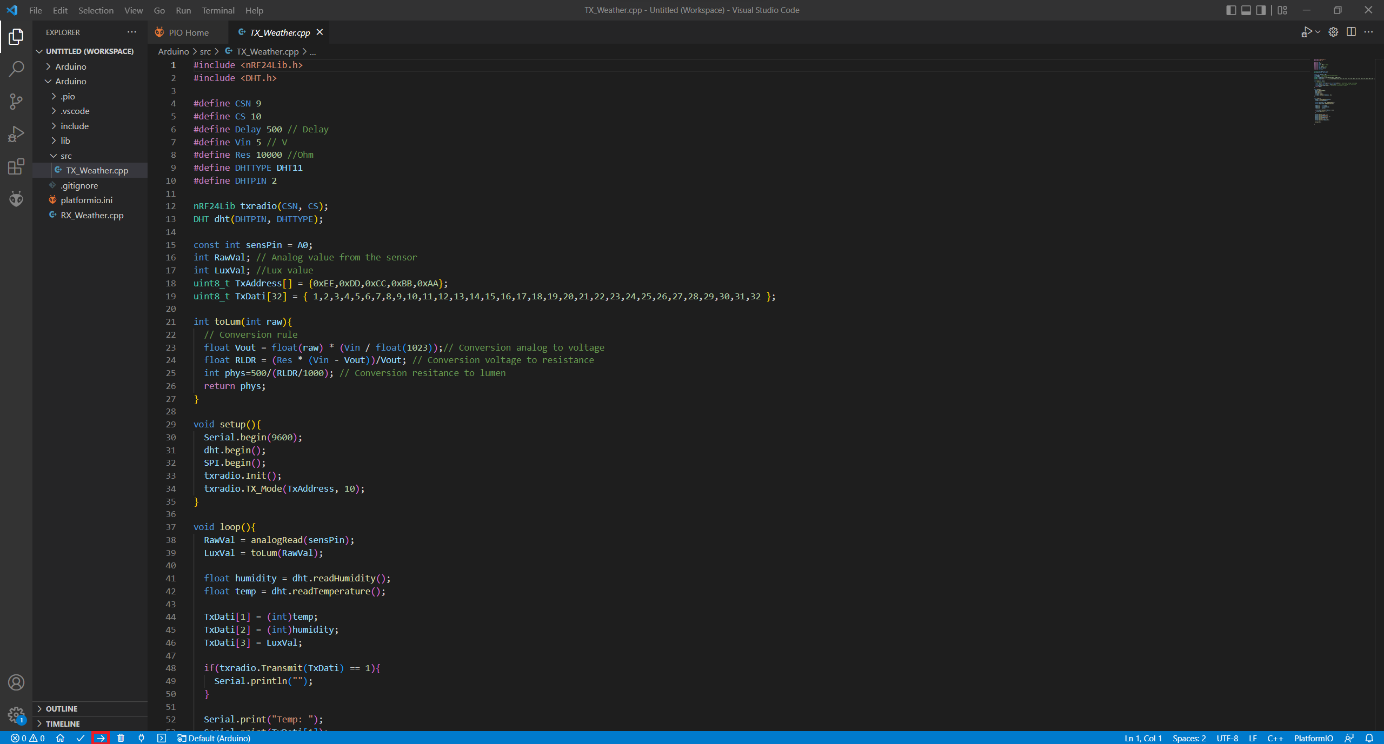
*10. att.*

1. Atvienojiet klienta mikro kontrolieri no datora.
2. Pievienojiet ar *USB* kabeli datoram centrālo mikro kontrolieru kuru saslēdzāt 2. solī.
3. Iekš *Visual Studio Code* apmainiet “src” mapīte failu TX\_Weather pret failu RX\_Weather.cpp. Skatīt 11.attēlu.



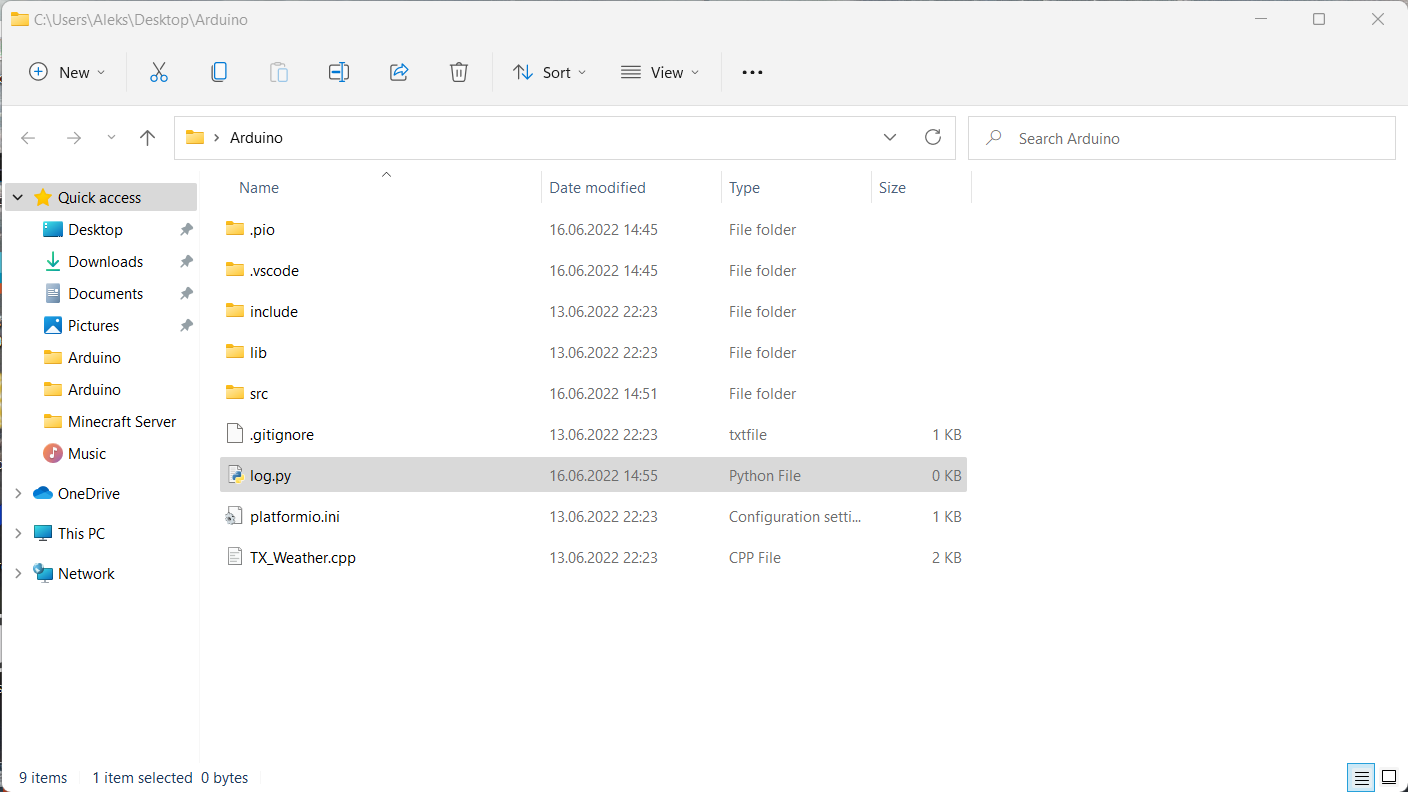
*11. att.*

1. Iekš *Visual Studio Code* spiediet pogu *“Upload”* un gaidiet, kamēr process būs pabeigts. Skatīt 12. attēlu.



*12. att.*

1. Iepriekš uz darbvirsmas izvilktajā mapītē veiciet ar peles kreiso pogu dubultklikšķi uz *“log.py”*. Skatīt 13.attēlu.



*13. att.*

1. Pievienojiet klienta mikro kontrolieri pie 9V strāvas avota.
2. Dati tiks ievākti ar klienta mikro kontrolieri un pārraidīti uz centrālo mikro kontrolieri, dators šos datus ievietos datubāzē, kur tos varēs apskatīt.

# **DATUBĀZES TABULA DATU GLABĀŠANAI**

Programmas nodrošināšanai nav nepieciešama liela datubāze, tiek izmantota viena tabula ievākto datu informācijas glabāšanai (skat. 2. tabulu). Datu bāzei tiek izmantota *MongoDB.*

2. tabula Lietotāja informācijas glabāšana

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Scheme** |
| id | INT | “id” INT NOT NULL PRIMARY KEY AUTOINCREMENT |
| time\_date | TIMESTAMP | “TIMEDATE” TIMESTAMP NOT NULL |
| temperature | DOUBLE | “temperature” DOUBLE NOT NULL |
| humidity | INT | “humidity” INT NOT NULL |
| lumens | INT | “lumens” INT NOT NULL |

# **SECINĀJUMI**

Izstrādājot dokumentāciju un darba praktisko daļu esmu ieguvis pieredzi dokumentācijas veidošanā un uzlabojis iemaņas *C / C++* programmēšanas valodā. Ar šo darbu esmu, arī papildinājis savu *GitHub* krātuvi, kas nākotnē dos pozitīvu iespaidu par manām spējām darba devējiem.

Nākotnē šo programmatūru varētu uzlabot, pievienojot funkcionalitāti, ka iegūtie dati tiek attēloti grafiski. Vēl noderīga funkcija būtu datu aplūkošana attālināti no jeb kurienes un integrācija ar mājas automatizāciju.

## Izmantoto informācijas avotu saraksts

1. nRF24L01+ Single Chip 2.4GHz Transceiver Preliminary Product Specification v1.0 [Datu lapa] Pieejams: <https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf> [Skatīts: 09.06.2022]
2. Arduino UNO R3 Datasheet [Datu lapa] Pieejams: <https://docs.arduino.cc/static/9982a40d35bb00ec42c0869106c40a0a/A000066-datasheet.pdf> [Skatīts: 09.06.2022]
3. DHT 22 Humidity & Temperature Sensor [Datu lapa] Pieejams: <https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf> [Skatīts: 09.06.2022]
4. Serial Peripheral Interface [Tiešsaistes resurss] Pieejams: <https://en.wikipedia.org/wiki/Serial_Peripheral_Interface> [Skatīts: 09.06.2022]

# **PIELIKUMI**

1. Pielikums “nRF24Lib.h”

#include <Arduino.h>

#include <SPI.h>

/\* Registri\*/

#define CONFIG 0x00

#define EN\_AA 0x01

#define EN\_RXADDR 0x02

#define SETUP\_AW 0x03

#define SETUP\_RETR 0x04

#define RF\_CH 0x05

#define RF\_SETUP 0x06

#define STATUS 0x07

#define OBSERVE\_TX 0x08

#define CD 0x09

#define RX\_ADDR\_P0 0x0A

#define RX\_ADDR\_P1 0x0B

#define RX\_ADDR\_P2 0x0C

#define RX\_ADDR\_P3 0x0D

#define RX\_ADDR\_P4 0x0E

#define RX\_ADDR\_P5 0x0F

#define TX\_ADDR 0x10

#define RX\_PW\_P0 0x11

#define RX\_PW\_P1 0x12

#define RX\_PW\_P2 0x13

#define RX\_PW\_P3 0x14

#define RX\_PW\_P4 0x15

#define RX\_PW\_P5 0x16

#define FIFO\_STATUS 0x17

#define DYNPD 0x1C

#define FEATURE 0x1D

/\* Komandas \*/

#define R\_REGISTER 0x00

#define W\_REGISTER 0x20

#define REGISTER\_MASK 0x1F

#define ACTIVATE 0x50

#define R\_RX\_PL\_WID 0x60

#define R\_RX\_PAYLOAD 0x61

#define W\_TX\_PAYLOAD 0xA0

#define W\_ACK\_PAYLOAD 0xA8

#define FLUSH\_TX 0xE1

#define FLUSH\_RX 0xE2

#define REUSE\_TX\_PL 0xE3

#define NOP 0xFF

class nRF24Lib{

public:

uint8\_t CSNpin;

uint8\_t CSpin;

/\*------------------------------------------------- nRF24Lib -----

| Konstruktors nRF24Lib

|

| Pielietojums: Izmanto, lai iestatitu CSN un CS pin, kas tiks izmantots biblioteka velak.

|

| Parametri:

| uint8\_t CSN (IN) -- Saja parametra ievada Arduino Chip Select Not kontakta numuru

| uint8\_t CS (IN) -- Saja parametra ievada Arduino Chip Select kontakta numuru

|

\*-------------------------------------------------------------------\*/

nRF24Lib(uint8\_t CSN, uint8\_t CS){

CSNpin = CSN;

CSpin = CS;

};

// Seit ir aprakstitas visas bibliotekas funkcijas

// un funkciju parametri

void CS\_Select(void);

void CS\_UnSelect(void);

void CE\_Enable(void);

void CE\_Disable(void);

void WriteRegister(uint8\_t reg, uint8\_t data);

void WriteRegisterMulti(uint8\_t reg, uint8\_t \* data, int size);

uint8\_t ReadRegister(uint8\_t reg);

void ReadRegisterMulti(uint8\_t reg, uint8\_t \* data, int size);

void SendCommand(uint8\_t cmd);

void Init(void);

void TX\_Mode(uint8\_t \*address, uint8\_t ch);

uint8\_t Transmit(uint8\_t \*dati);

void RX\_Mode(uint8\_t \*address, uint8\_t ch);

uint8\_t IsDataAvailable(int pipenr);

void Receive(uint8\_t \*data);

};

1. Pielikums “nRF24Lib.cpp”

#include "nRF24Lib.h"

// SPI iestatijumi 4MHz, Most Significant Byte First un CPOL 0, CPHA 0

SPISettings SPIrfsett(4000000, MSBFIRST, SPI\_MODE0);

/\*------------------------------------------------- CS\_Select -----

| Funkcija CS\_Select

|

| Pielietojums: Iestata Chip Select pin uz zemo limeni

|

\*-------------------------------------------------------------------\*/

void nRF24Lib::CS\_Select(void){

digitalWrite(CSpin, LOW);

}

/\*------------------------------------------------- CS\_UnSelect -----

| Funkcija CS\_UnSelect

|

| Pielietojums: Iestata Chip Select pin uz augsto limeni

|

\*-------------------------------------------------------------------\*/

void nRF24Lib::CS\_UnSelect(void){

digitalWrite(CSpin, HIGH);

}

/\*------------------------------------------------- CE\_Enable -----

| Funkcija CE\_Enable

|

| Pielietojums: Iestata Chip Enable pin uz augsto limeni

|

\*-------------------------------------------------------------------\*/

void nRF24Lib::CE\_Enable(void){

digitalWrite(CSNpin, HIGH);

}

/\*------------------------------------------------- CE\_Disable -----

| Funkcija CE\_Disable

|

| Pielietojums: Iestata Chip Enable pin uz zemo limeni

|

\*-------------------------------------------------------------------\*/

void nRF24Lib::CE\_Disable(void){

digitalWrite(CSNpin, LOW);

}

/\*------------------------------------------------- WriteRegister -----

| Funkcija WriteRegister

|

| Pielietojums: Si funkcija lauj ierakstīt nRF24L01 registra datus.

|

| Parametri:

| uint8\_t reg (IN) -- saja parametra norada registra adresi

| uint8\_t data (IN) -- saja parametra ievada velamo vertibu

| ko iestatit registra

|

\*-------------------------------------------------------------------\*/

void nRF24Lib::WriteRegister(uint8\_t reg, uint8\_t data){

uint8\_t buf[2];

buf[1] = reg|1<<5;

buf[2] = data;

SPI.beginTransaction(SPIrfsett);

CS\_Select();

SPI.transfer(buf[1]);

SPI.transfer(buf[2]);

CS\_UnSelect();

SPI.endTransaction();

}

/\*------------------------------------------------- WriteRegisterMulti -----

| Funkcija WriteRegisterMulti

|

| Pielietojums: Si funkcija lauj ierakstīt nRF24L01 registra datu masivu.

|

| Parametri:

| uint8\_t reg (IN) -- saja parametra norada registra adresi

| uint8\_t \* data (IN) -- saja parametra ievada velamo vertibu masivu

| ko ierakstit registra

| int size (IN) -- Saja parametra janorada rakstamo datu masiva garumu

|

\*-------------------------------------------------------------------\*/

void nRF24Lib::WriteRegisterMulti(uint8\_t reg, uint8\_t \* data, int size){

uint8\_t buf[2];

buf[1] = reg|1<<5;

SPI.beginTransaction(SPIrfsett);

CS\_Select();

SPI.transfer(buf[1]);

for(int i = 0; i < size; i++)

{

SPI.transfer(data[i]);

}

CS\_UnSelect();

SPI.endTransaction();

}

/\*------------------------------------------------- ReadRegister -----

| Funkcija ReadRegister

|

| Pielietojums: Si funkcija lauj nolasit nRF24L01 registra vertibu.

|

| Atgriez: Si funkcija atgriez nolasama registra vertibu.

|

| Parametri:

| uint8\_t reg (IN) -- saja parametra norada nolasama registra adresi

|

\*-------------------------------------------------------------------\*/

uint8\_t nRF24Lib::ReadRegister(uint8\_t reg){

uint8\_t dati;

SPI.beginTransaction(SPIrfsett);

CS\_Select();

SPI.transfer(reg);

dati = SPI.transfer(0);

CS\_UnSelect();

SPI.endTransaction();

//Serial.println(dati);

return dati;

}

/\*------------------------------------------------- ReadRegisterMulti -----

| Funkcija ReadRegisterMulti

|

| Pielietojums: Si funkcija lauj nolasit nRF24L01 registra vertibas un ielasit to masiva.

|

| Parametri:

| uint8\_t reg (IN) -- saja parametra norada nolasama registra adresi

| uint8\_t \*data (OUT) -- saja parametra norada raditaju uz masivu kur tiks ielasiti dati no registra

| int size (IN) -- saja parametra norada masiva (kur tiks ielasiti dati) lielumu

|

\*-------------------------------------------------------------------\*/

void nRF24Lib::ReadRegisterMulti(uint8\_t reg, uint8\_t \*data, int size){

SPI.beginTransaction(SPIrfsett);

CS\_Select();

SPI.transfer(&reg, 1);

\*data = SPI.transfer(0);

CS\_UnSelect();

SPI.endTransaction();

}

/\*------------------------------------------------- SendCommand -----

| Funkcija SendCommand

|

| Pielietojums: Si funkcija nosuta komandu uz nRF24L01 moduli

|

| Parametri:

| uint8\_t cmd (IN) -- Nosutama komanda

|

\*-------------------------------------------------------------------\*/

void nRF24Lib::SendCommand(uint8\_t cmd){

SPI.beginTransaction(SPIrfsett);

CS\_Select();

SPI.transfer(&cmd, 1);

CS\_UnSelect();

SPI.endTransaction();

}

/\*------------------------------------------------- Init -----

| Funkcija Init

|

| Pielietojums: Funkcija sagatavo nRF24L01 bezvadu moduli darbībai.

| Vispirms konfigurācija tiek nodzēsta iestatot visas vērtības uz 0.

| Pēc tam tiek izslēgta Auto-Acknowledgment funkcija, izslēgtas visas

| datu saņemšanas līnijas, saņemšanas/sūtīšanas līniju platums tiek iestatīts uz 5 baitiem,

| Auto-Retransmit tiek iestatīts uz 250 mikrosekundēm, moduļa frekvence tiek iestatīta uz

| 2400 MHz un ātrums uz 2Mbps. Funkcija neatgriež vērtību un nepieprasa parametrus.

|

\*-------------------------------------------------------------------\*/

void nRF24Lib::Init(void){

CE\_Disable();

WriteRegister(CONFIG, 0);

WriteRegister(EN\_AA, 0);

WriteRegister(EN\_RXADDR, 0);

WriteRegister(SETUP\_AW, 0x03);

WriteRegister(SETUP\_RETR, 0);

WriteRegister(RF\_CH, 0);

WriteRegister(RF\_SETUP, 0x0E);

CE\_Enable();

}

/\*------------------------------------------------- TX\_Mode -----

| Funkcija TX\_Mode

|

| Pielietojums: Si funkcija iestata nRF24L01 bezvadu moduli sutisanas rezima

|

| Parametri:

| uint8\_t \*address (IN) -- Saja parametra ievada raditaju uz masivu kurs satur sanemeja adresi

| uint8\_t ch (IN) -- Saja parametra ievada kanalu uz kurieni sutit datus (Frekvence ir 2400MHz + ch)

|

\*-------------------------------------------------------------------\*/

void nRF24Lib::TX\_Mode(uint8\_t \*address, uint8\_t ch){

CE\_Disable();

WriteRegister(RF\_CH, ch);

WriteRegisterMulti(TX\_ADDR, address, 5);

uint8\_t cfg = ReadRegister(CONFIG);

cfg = cfg | (1<<1);

WriteRegister(CONFIG, cfg);

CE\_Enable();

}

/\*------------------------------------------------- Transmit -----

| Funkcija Transmit

|

| Pielietojums: Si funkcija nosuta datus izmantojot nRF24L01

|

| Parametri:

| uint8\_t \*dati (IN) -- Saja parametra ievada raditaju uz masivu kurs satur sutamos datus.

| Sim masivam ir jabut 32 garam.

|

\*-------------------------------------------------------------------\*/

uint8\_t nRF24Lib::Transmit(uint8\_t \*dati){

//uint8\_t cmdsend = 0;

SPI.beginTransaction(SPIrfsett);

CS\_Select();

//cmdsend = W\_TX\_PAYLOAD;

SPI.transfer(W\_TX\_PAYLOAD);

for(int i = 0; i < 32; i++)

{

SPI.transfer(dati[i]);

}

CS\_UnSelect();

SPI.endTransaction();

delay(1);

uint8\_t fifostatus;

fifostatus = ReadRegister(FIFO\_STATUS);

if ((fifostatus&(1<<4))

&& (!(fifostatus&(1<<3)))){

//cmdsend = FLUSH\_TX;

SendCommand(FLUSH\_TX);

return 1;

}

return 0;

}

/\*------------------------------------------------- RX\_Mode -----

| Funkcija RX\_Mode

|

| Pielietojums: Si funkcija iestata nRF24L01 bezvadu moduli sanemsanas rezima

|

| Parametri:

| uint8\_t \*address (IN) -- Saja parametra ievada raditaju uz masivu kurs satur sutitaja adresi

| uint8\_t ch (IN) -- Saja parametra ievada kanalu uz ka gaidit sanemamos datus (Frekvence ir 2400MHz + ch)

|

\*-------------------------------------------------------------------\*/

void nRF24Lib::RX\_Mode(uint8\_t \*address, uint8\_t ch){

CE\_Disable();

WriteRegister(RF\_CH, ch);

uint8\_t enrxaddr = ReadRegister(EN\_RXADDR);

enrxaddr = enrxaddr | (1<<1);

WriteRegister(EN\_RXADDR, enrxaddr);

WriteRegisterMulti(RX\_ADDR\_P1, address, 5);

WriteRegister(RX\_PW\_P1, 32);

uint8\_t cfg = ReadRegister(CONFIG);

cfg = cfg | (1<<1) | (1<<0);

WriteRegister(CONFIG, cfg);

CE\_Enable();

}

/\*------------------------------------------------- IsDataAvailable -----

| Funkcija IsDataAvailable

|

| Pielietojums: Si funkcija iestata nRF24L01 bezvadu moduli sanemsanas rezima

|

| Atgriez: Funkcija atgriez 1 vai 0 atkariba no ta vai dati ir pieejami sanemsanai

|

| Parametri:

| int pipenr (IN) -- Saja parametra ievada datu linijas numuru, kuru parbaudit uz sanemajamiem datiem.

|

\*-------------------------------------------------------------------\*/

uint8\_t nRF24Lib::IsDataAvailable(int pipenr){

uint8\_t status = ReadRegister(STATUS);

if((status&(1<<6))

&&(status&(pipenr<<1))){

WriteRegister(STATUS, (1<<6));

return 1;

}

return 0;

}

/\*------------------------------------------------- Transmit -----

| Funkcija Transmit

|

| Pielietojums: Si funkcija sanem datus izmantojot nRF24L01

|

| Parametri:

| uint8\_t \*data (OUT) -- Saja parametra ievada raditaju uz masivu kur sanemt datus.

| Sim masivam ir jabut 32 garam.

|

\*-------------------------------------------------------------------\*/

void nRF24Lib::Receive(uint8\_t \*data){

//uint8\_t cmdsend = 0;

SPI.beginTransaction(SPIrfsett);

CS\_Select();

//cmdsend = R\_RX\_PAYLOAD;

SPI.transfer(R\_RX\_PAYLOAD);

for(int i = 0; i < 32; i++)

{

data[i] = SPI.transfer(0);

}

CS\_UnSelect();

SPI.endTransaction();

delay(1);

//cmdsend = FLUSH\_RX;

SendCommand(FLUSH\_RX);

}

1. Pielikums “RX\_Weather.cpp”

#include <nRF24Lib.h>

 #define CSN 9

#define CS 10

nRF24Lib rxradio(CSN, CS);

uint8\_t RxAddress[] = {0xEE,0xDD,0xCC,0xBB,0xAA};

uint8\_t RxDati[32];

void setup(){

  Serial.begin(9600);

  SPI.begin();

  Serial.println("SPI BEGIN");

  rxradio.Init();

  rxradio.RX\_Mode(RxAddress, 10);

  Serial.println("RX BEGIN");

}

void loop(){

  if (rxradio.IsDataAvailable(1) == 1)

    {

      rxradio.Receive(RxDati);

      for(int i = 0; i < sizeof(RxDati); i++)

      {

        Serial.print(RxDati[i]);

        Serial.print(" ");

      }

      Serial.println("");

    }

  delay(1000);

}

1. Pielikums “TX\_Weather.cpp”

#include <nRF24Lib.h>

#include <DHT.h>

#define CSN 9

#define CS 10

#define Delay 500 // Delay

#define Vin 5 // V

#define Res 10000 //Ohm

#define DHTTYPE DHT11

#define DHTPIN 2

nRF24Lib txradio(CSN, CS);

DHT dht(DHTPIN, DHTTYPE);

const int sensPin = A0;

int RawVal; // Analog value from the sensor

int LuxVal; //Lux value

uint8\_t TxAddress[] = {0xEE,0xDD,0xCC,0xBB,0xAA};

uint8\_t TxDati[32] = { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32 };

int toLum(int raw){

  // Conversion rule

  float Vout = float(raw) \* (Vin / float(1023));// Conversion analog to voltage

  float RLDR = (Res \* (Vin - Vout))/Vout; // Conversion voltage to resistance

  int phys=500/(RLDR/1000); // Conversion resitance to lumen

  return phys;

}

void setup(){

  Serial.begin(9600);

  dht.begin();

  SPI.begin();

  txradio.Init();

  txradio.TX\_Mode(TxAddress, 10);

}

void loop(){

  RawVal = analogRead(sensPin);

  LuxVal = toLum(RawVal);

  float humidity = dht.readHumidity();

  float temp = dht.readTemperature();

  TxDati[1] = (int)temp;

  TxDati[2] = (int)humidity;

  TxDati[3] = LuxVal;

  if(txradio.Transmit(TxDati) == 1){

    Serial.println("");

  }

  Serial.print("Temp: ");

  Serial.print(TxDati[1]);

  Serial.print(" Humidity: ");

  Serial.print(TxDati[2]);

  Serial.print(" Lux: ");

  Serial.println(TxDati[3]);

  delay(300);

}