Udaljeno upravljanje farme primjenom Interneta stvari

Međimorec, Karlo

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Karlo Međimorec

UDALJENO UPRAVLJANJE FARME PRIMJENOM INTERNETA STVARI

ZAVRŠNI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Karlo Međimorec

Matični broj: 0016147525

Studij: Sveučilišni prijediplomski studij Informacijski/Poslovni sustavi

UDALJENO UPRAVLJANJE FARME PRIMJENOM INTERNETA STVARI

ZAVRŠNI RAD

Mentor/Mentorica:

Prof. Dr. Sc. Ivan Magdalenić

Varaždin, kolovoz 2023.

Karlo Međimorec

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Stvaranje sustava za udaljeno nadziranje i upravljanje farmom. Za izradu sustava potrebno je istražiti trendove u IoT sustavima koji se odnose na izradu ovog rada. Prema istraženome treba napraviti rješenje sustava: definirati koji uređaji će se koristiti, koji programski jezici će se koristiti na tim uređajima, koja razvojna okruženja će se koristiti za pisanje programskog koda, koji komunikacijski standardi će se koristiti. Nakon definiranja rješenja sustava potrebno ga je implementirati. U fazi implementacije radi se na spajanju elektroničkih uređaja, pisanje programskog koda za ugrađene sustave, korištenje raznih programskih alata za pisanje i izvođenje programskog koda, izrada aplikacije u .NET okviru, rad sa bazom podataka, rad sa IoT standardima.

Ključne riječi: IoT; Ugrađeni sustav; Pametni sustav; MQTT; ESP32; Raspberry Pi; Broker; .NET

Sadržaj

1. Uvod

Tema završnog rada je sustav koji će motriti i upravljati određenim dijelovima farme. Sustav prati temperaturu, vlagu i svijetlost, te ima mogućnost paljenja i gašenja svijetla. Sustav ima mogućnost prikazivanja trenutnih parametara koje prati (maksimalno zaostajanje od 5 minuta) i prikazivanje parametara koji su pohranjeni u bazi podataka (zadnjih 60 dana.) Sustav se sastoji od 2 ESP32 uređaja, Raspberry Pi računala i .NET aplikacije. ESP32 uređaji prate parametre farme te ih šalju na MQTT poslužitelj. Raspberry Pi dohvaća podatke sa tema na koje ih šalju ESP32 uređaji. Primljene podatke akumulira, te jednom dnevno pohranjuje prosječnu vrijednost akumuliranih podataka u lokalnu bazu podataka (MariaDB). .NET aplikacija služi kao grafičko korisničko sučelje, preko kojeg korisnik može vidjeti trenutno i povijesno stanje svoje farme. Aplikacija za komunikaciju koristi MQTT, te je spojena na teme koje koriste i ESP32 uređaji i Raspberry Pi kako bi se omogućio uvid u trenutne parametre i povijesne parametre. Osim što sluša teme, aplikacija i šalje podatke na posebnu temu kako bi se omogućilo paljenje/gašenje svijetla.

Motivacija kod odabira ove teme je želja da se napravi završni rad koji će se koristiti u stvarnom svijetu, te koji će pomoći nekome kod svakodnevnog posla. Kroz izradu ovog rada prikazana su znanja koja su stečena tijekom edukacije na fakultetu, te koja znanja su najviše utjecala na mene tijekom učenja.

2. Metode i tehnike rada

U ovom poglavlju biti će objašnjen način na koji je provedeno istraživanje svega vezanog uz izradu rada. Osim toga biti će navedena i objašnjena sva integrirana razvojna okruženja koja su se koristila kod izradu programa i aplikacija. Biti će navedeni i svi programski jezici koji se koriste na ugrađenim uređajima, Raspberry Pi računalu i računalu krajnjeg korisnika, te svo sklopovlje koje se koristi za realizaciju rada.

2.1. Istraživanje

Istraživanje o svim korištenim tehnologijama je napravljeno uz pomoć interneta. Za istraživanje sklopovlja korištene su službene stranice proizvođača, dok su za istraživanje programskih jezika korišteni razni forumi i stranice koje su namijenjene za pomoć pri pisanju programskog koda.

2.2. Programski alati

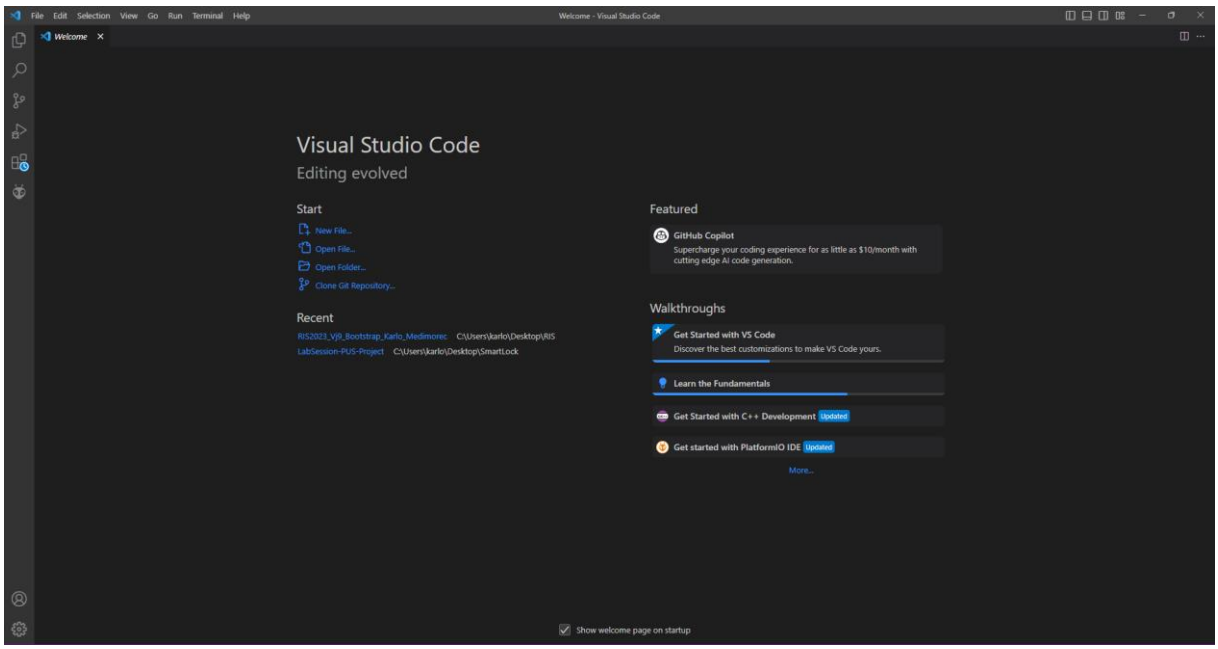
U radu su korišteni razni programski alati za pisanje, uređivanje i izvođenje koda, te programski alat za izradu i kontrolu baze podataka. U ovom podpoglavlju biti će objašnjeni sljedeći programski alati: Visual Studio Code (Platformio), Visual Studio, Thonny i MariaDB.

2.2.1. Visual Studio Code

Za pisanje koda koji se nalazi na ESP32 uređajima, korišten je Visual Studio Code. InfoWorld opisuje Visual Studio Code na sljedeći način:

> Visual Studio Code besplatan je, lagan, ali moćan uređivač izvornog koda koji radi na vašem stolnom računalu i na webu i dostupan je za Windows, macOS, Linux i Raspberry Pi OS. Dolazi s ugrađenom podrškom za JavaScript, TypeScript i Node.js te ima bogat ekosustav proširenja za druge programske jezike (kao što su C++, C#, Java, Python, PHP i Go), vremena izvođenja (kao što je .NET i Unity), okruženja (kao što su Docker i Kubernetes) i oblake (kao što su Amazon Web Services, Microsoft Azure i Google Cloud Platform) (Heller, 2022).

Visual Studio Code izabran je zbog svog bogatog ekosustava proširenja, zbog kojeg je, između ostalih, podržana PlatformIO ekstenzija. PlatformIO je alat koji omogućuje pisanje programskog koda za ugrađene sustave, te stavljanje tog koda na ugrađene uređaje.



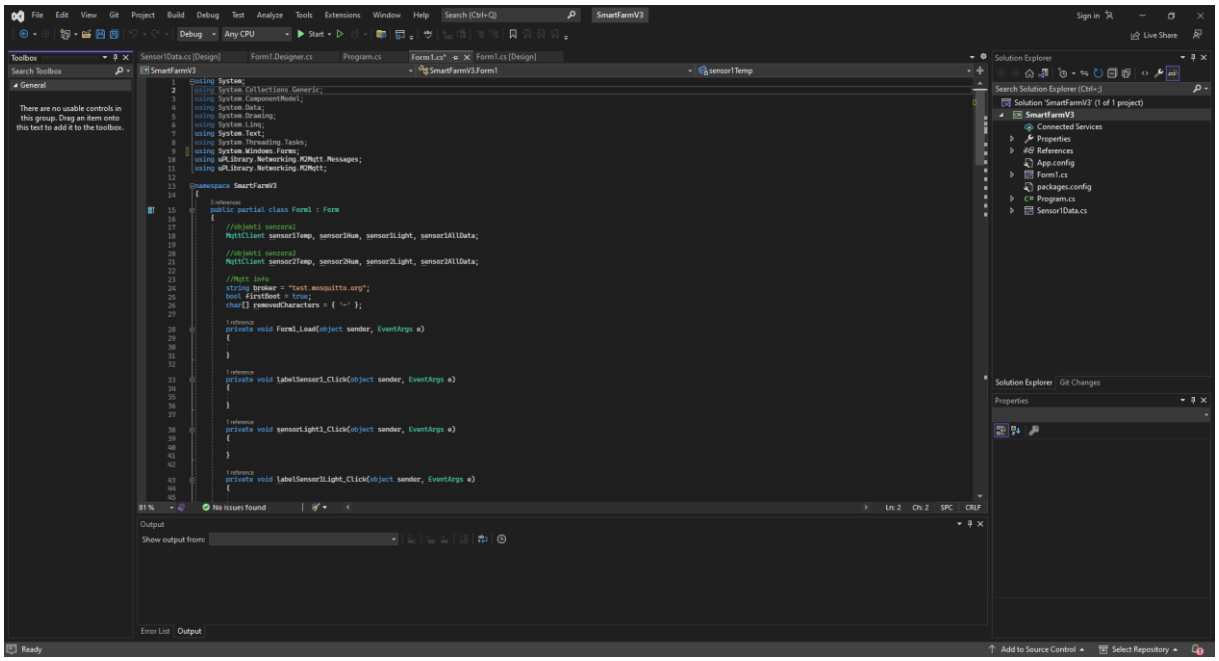
Slika 1: Visual Studio Code (Izvor: autor, 2023)

2.2.2. Visual Studio

Visual Studio korišten je za izradu .NET aplikacije koja služi kao grafičko korisničko sučelje. Prema InfoWorld-u Visual Studio je sljedeće:

> Visual Studio je Microsoftov vrhunski IDE za Windows i macOS. Uz Visual Studio možete razvijati, analizirati, ispravljati pogreške, testirati, surađivati na i implementirati svoj softver (Heller, 2022).

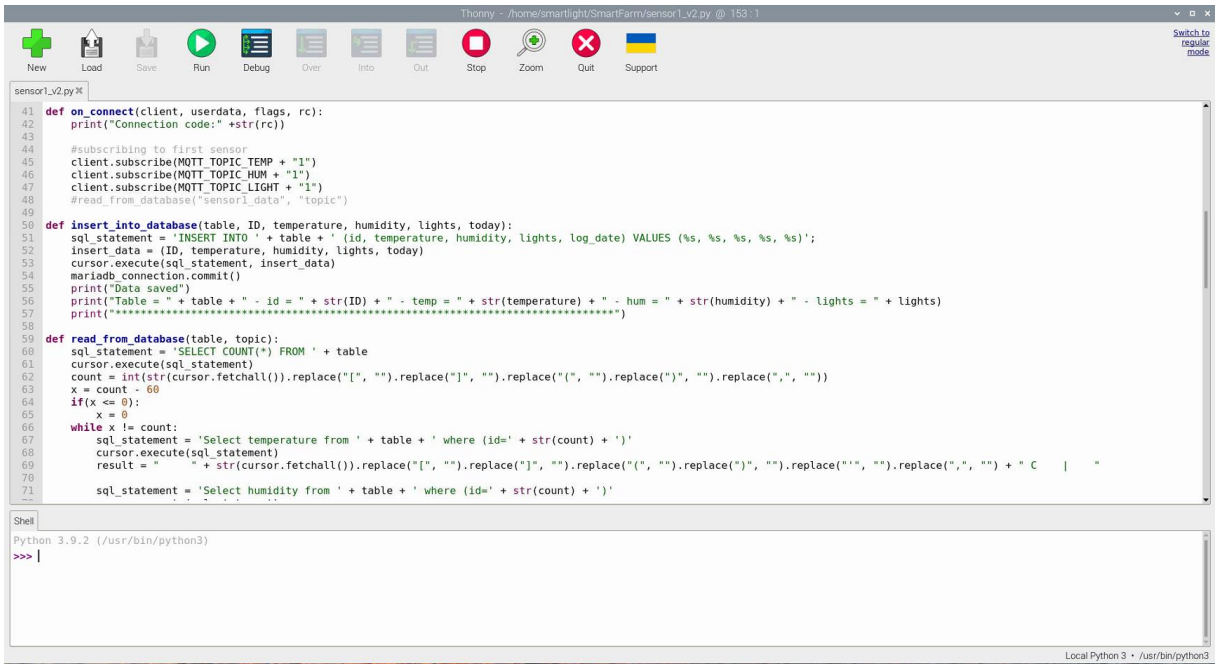
Ovaj IDE je izabran zbog toga što je najčešće korišten kod izradnje .NET aplikacije. Osim što je najčešće korišten, on omogućuje izgradnju aplikacije pomoću Windows Forms-a, što uvelike olakšava proces izgradnje aplikacije koja će se koristiti na Windows operativnom sustavu.



Slika 2: Visual Studio 2022 (Izvor: autor, 2023)

2.2.3. Thonny

Thony je vrlo jednostavno i besplatno integrirano razvojno okruženje otvorenog koda, koje je razvijeno za početnike. Omogućava pisanje i izvođenje python skripte na računalu. Zbog toga što je jednostavno, omogućuje samo osnovne radnje nad programskim kodom, a to su: isticanje sintakse, pronalaženje grešaka u sintaksi, pohrana koda, pokretanje koda, debugging koda i zaustavljanje koda. Ovaj IDE je, zbog svoje jednostavnosti, vrlo lagan zbog čega se lako može koristiti na računalima slabijih performansi. Zbog toga je odabran za ovaj završni rad, jer se koristio za pisanje Python skripte na Raspberry Pi računalu.



Slika 3: Thonny (Izvor: autor, 2023)

2.2.4. MariaDB

MariaDB je korištena kao baza podataka unutar koje se spremaju prikupljeni podaci sa senzora. Ova baza podatak je vrlo laka i jednostavna za korištenje relacijska baza podataka. Instalira se preko naredbenog retka, te se unutar njega i koristi pomoću SQL naredbi. Postoje grafička korisnička sučelja, koja su napravljena od strane zajednice. U članku builtin-a, MariaDB je opisana kao:

> MariaDB je brza, skalabilna i podržava više mehanizama za pohranu od MySQL-a. Slično MySQL-u, MariaDB podržava vanjske dodatke, što znači da možete proširiti bazu podataka i primijeniti je u više slučajeva upotrebe, kao što su e-trgovina, skladištenje podataka i aplikacije za bilježenje (Jalli, 2022).

2.3. Programski jezici

U izradi rada korišteno je više programskih jezika. Više programskih jezika je korišteno kako bi se prilagodilo svakom korištenom sklopovlju. U ovom podpoglavlju su objašnjeni sljedeći programski jezici: C# (.NET aplikacija), C++ (ESP32) i Python (Raspberry Pi).

2.3.1. C#

C# je programski jezik koji je korišten za izradi .NET aplikacije, tj. aplikacije koja služi kao grafičko korisničko sučelje. Prema builtin-u C# je:

> C# je Microsoftov programski jezik. C# kombinira računalnu snagu C++ i jednostavnost Visual Basica, Microsoftovog programskog jezika i okruženja vođenog događajima. C# se temelji na C++, ima značajke slične Javi i aplikacije u mnogim područjima razvoja softvera. Na tržištu rada postoji velika potražnja za C# programerima. Programeri mogu koristiti C# u mnogim područjima razvoja softvera, uključujući razvoj igara i aplikacija (Jalli, 2022).

Pošto je aplikacija zamišljena tako da se izvodi na računalu koje pokreče operacijski sustav Windows, odabran je .NET razvojni okvir, a samim time i C# programski jezik.

2.3.2. C++

C++ je programski jezik koji je korišten na ESP32 uređajima. Ovaj jezik je odabran zbog toga što ga ESP32 podržava bez ikakve potrebe za konfiguracijom ili instalacijom dodatnih paketa. Geeksforgeeks govori da je C++:

> C++ je programski jezik opće namjene koji je razvijen kao unapređenje jezika C za uključivanje objektno orijentirane paradigme. To je imperativ i kompilirani jezik… Sve u svemu, C++ je moćan i svestran programski jezik koji se naširoko koristi za niz aplikacija i dobro je prikladan i za sistemsko programiranje niske razine i za razvoj aplikacija na visokoj razini (Lenka, 2023).

2.3.3. Python

Python je korišten za sav kod koji je napisan na Raspberry Pi računalu. Ovaj jezik je odabran za Raspberry Pi zbog toga što je sam proizvođač ovog računala odabrao Python kao glavni jezik za razvoj. Osim toga python je odabran i zbog toga što se kroz preddiplomski studij student ne susreće sa njime, stoga je dobro naučiti nešto novo. Organizacija Python ga opisuje kao:

> Python je interpretirani, objektno orijentirani programski jezik visoke razine s dinamičkom semantikom. Njegove visoke razine ugrađenih podatkovnih struktura, u kombinaciji s dinamičkim tipkanjem i dinamičkim vezanjem, čine ga vrlo atraktivnim za brzi razvoj aplikacija, kao i za

korištenje kao skriptni ili ljepljivi jezik za povezivanje postojećih komponenti. Pythonova jednostavna sintaksa koju je lako naučiti naglašava čitljivost i stoga smanjuje troškove održavanja programa. Python podržava module i pakete, što potiče modularnost programa i ponovnu upotrebu koda („*What is Python? Executive Summary*“, 2023).

2.4. Sklopovlje

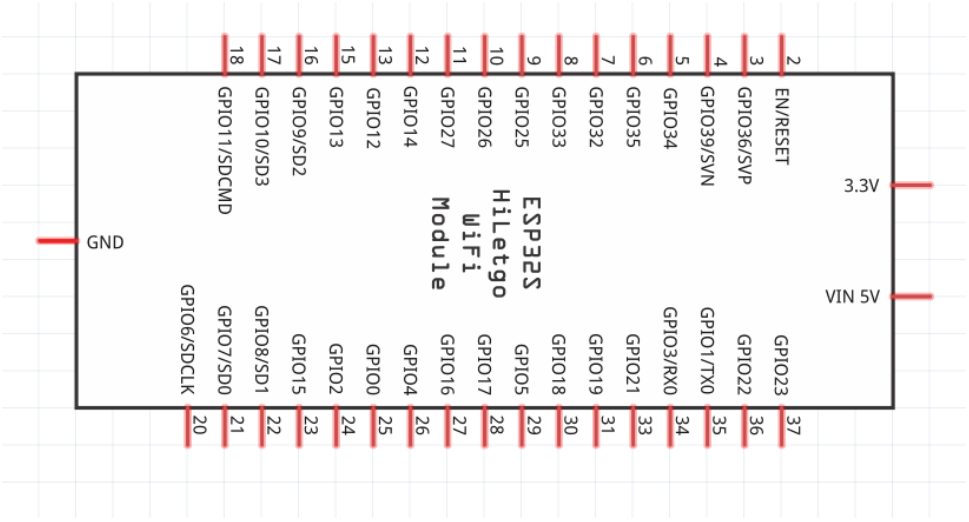
U izradi rada korišteno je više uređaja i više vrsta senzora. Kroz ovo podpoglavlje biti će objašnjeno sljedeće sklopovlje: ESP32, periferni uređaji za ESP32 (senzori), Raspberry Pi računalo i relay (spojen sa Raspberry Pi računalom).

2.4.1. ESP32

ESP32 je ugrađeni uređaj koji se sastoji od više međusobno povezanih SOC(System on a Chip) modula. Karakteriziraju ga niska cijena, velika modularnost i mala potrošnja električne energije. Ima osnovne mogućnosti kao što su izvođenje programskog koda, bežično spajanje na mrežu (WiFi, Bluetooth, LoRa…), svestrano periferno sučelje i vrlo sposobni sustav za upravljanje energijom.

U ovom radu služi kao senzor koji prati parametre farme. Stoga se ponaša kao edge uređaj koji se nalazi unutar same farme. Prikuplja parametre svakih 5 minuta te ih šalje na MQTT poslužitelj. Kako bi ih slao na MQTT poslužitelj povezan je na WiFi. Napajanje dobiva preko baterije kako bi se osigurao rad sustava ukoliko se dogodi neka nepogoda, kao što je nestanak struje.

Za ovaj projekt je odabran zbog svoje jednostavnosti, svestranosti i niske cijene samog uređaja te perifernih uređaja koji se koriste s njime. Osim samog ESP32, u ovom radu, koristi se ESP-WROOM-32 Expansion Board sa CH340C čipom. To znači da osim samog ESP32 SOC-a uređaj dolazi na pločici koja omogućuje, između ostalog, lako spajanje perifernih uređaja i baterije. Ukratko WROOM je razvojni komplet. CH340C je čip koji služi za pretvaranje USB-a na UART, tj. omogućuje lako prenašanje programskog koda sa računala na ESP32 preko USB kabla. Shematski prikaz ESP-WROOM-32 se nalazi na slici ispod.



Slika 4: ESP-WROOM-32 (Izvor: autor, 2023)

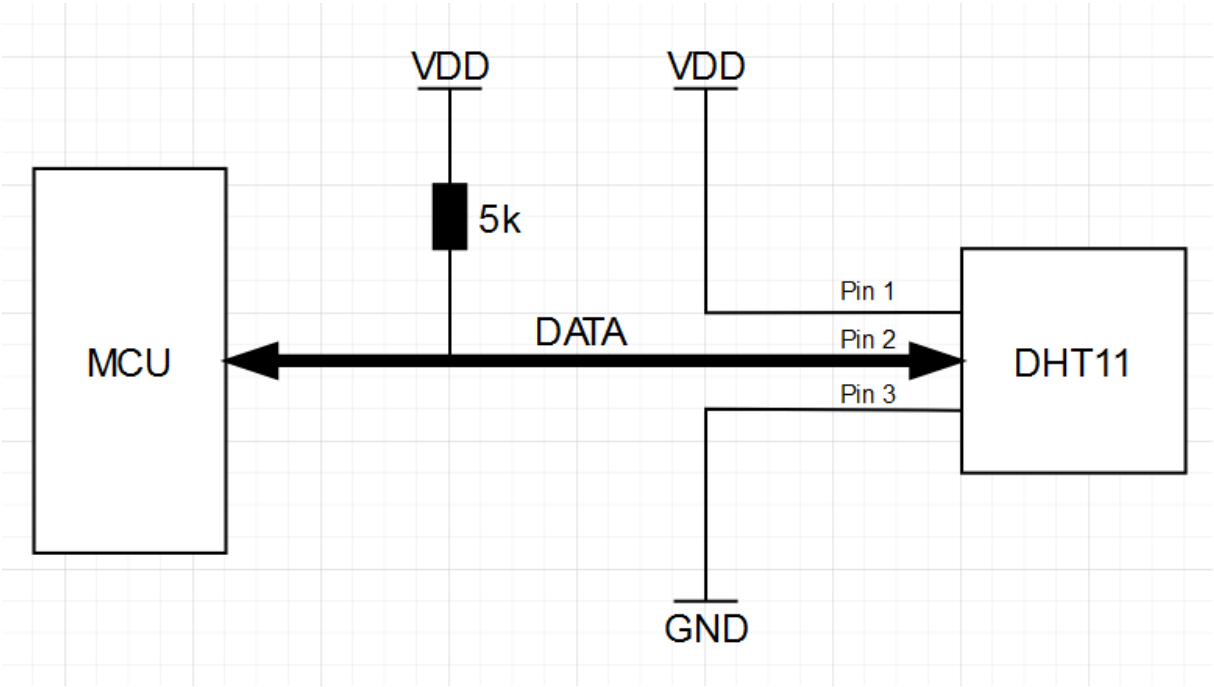
2.4.2. Periferni uređaji za ESP32

U ovom radu korišteni su sljedeći periferni uređaji: DHT11, LDR-LM393 i 0.96’ OLED ekran. U nastavku će biti detaljno opisani svi uređaji i prikazane njihove sheme.

2.4.2.1. DHT11

DHT11 je senzor koji služi za dohvaćanje trenutne temperature i vlage zraka, koje zatim pretvara u digitalni oblik i šalje do ESP32. Proizvođač Mouser u svojoj službenoj dokumentaciji opisuje ovaj senzor na sljedeći način:

> DHT11 senzor temperature i vlažnosti sadrži senzor temperature i vlažnosti kompleks s kalibriranim izlazom digitalnog signala. Korištenjem ekskluzivne akvizicije digitalnog signala tehnika i tehnologija senzora temperature i vlage, osigurava visoku pouzdanost i izvrsnu dugoročna stabilnost. Ovaj senzor uključuje komponentu otporničkog mjerenje vlažnosti i NTC komponentu za mjerenje temperature koje se spajaju na 8-bitni mikrokontroler visokih performansi. Nudi izvrsnu kvalitetu, brz odziv, zaštitu od smetnji i isplativost. (Mouser Electronics, n.d.)

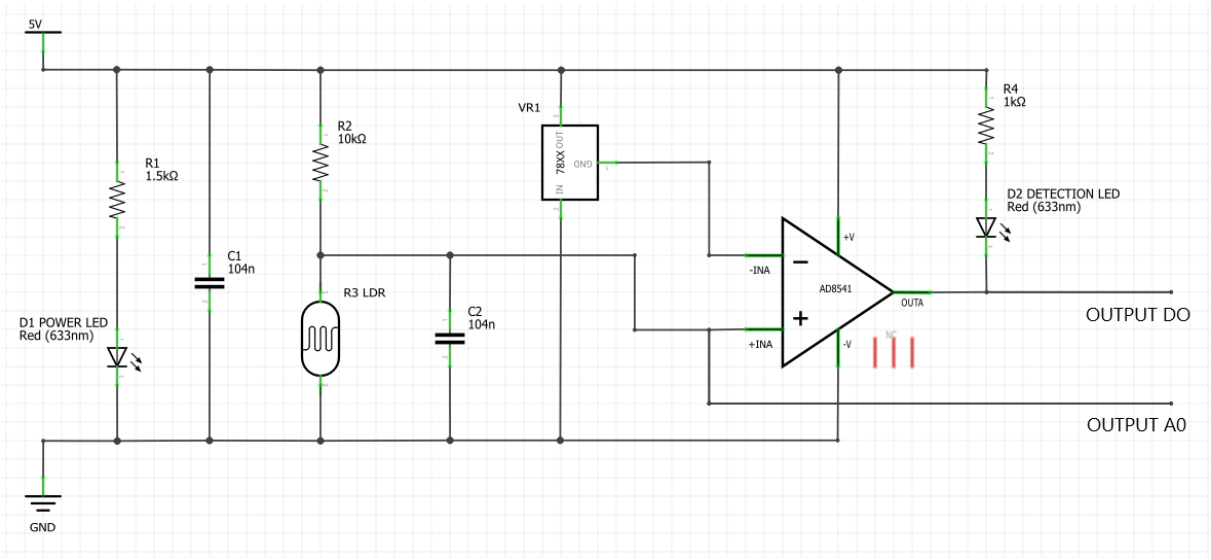


Slika 5: Shematski prikaz DHT11 senzora (Izvor: autor, 2023)

2.4.2.2. LDR-LM393

LDR-LM393 je senzor koji, u ovom radu, služi za promatranje razine svjetlosti unutar farme. Ovo je digitalna verzija senzora, što znači da vraća samo vrijednosti 0 ili 1. Radi tako da prati razinu svijetlosti, te je uspoređuje sa zadanom vrijednošću koju je zadao korisnik. Kada je razina svjetlosti iznad zadane razine vraća se vrijednost 0, obrnuto kada je razina svjetlosti ispod zadane razine vraća se vrijednost 1. Korisnik vrijednost zadaje fizičkim okretanjem potenciometra na samom senzoru, zbog čega se razina ne može zadati preko programskog koda.

Ovo je fotosenzitivan senzor koji radi na temelju LDR (Light Dependant Resistor), drugim riječima fotosenzitivnog otpornika. Radi tako da se otpornost otpornika (Ohm) smanjuje sa povećanjem intenziteta svijetlosti (Lux), što uzrokuje povećanjem vodljivosti elementa. Obrnuto smanjenjem intenziteta svijetlosti smanjuje se vodljivost elementa. LM393 je čip koji služi za uspoređivanje zadane vrijednosti i vodljivosti elementa. Na temelju toga ovaj uređaj vraća digitalnu vrijednost.



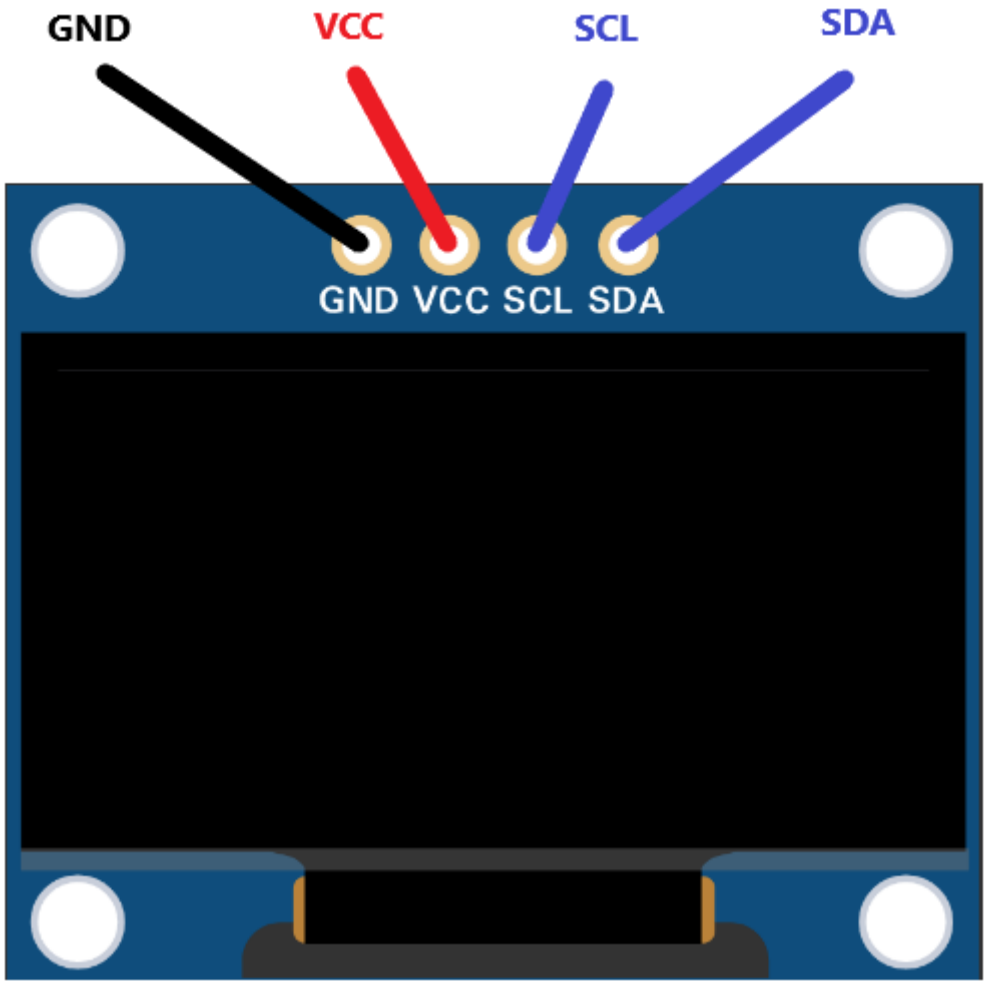
Slika 6: Shematski prikaz LDR senzora (Izvor: autor, 2023)

2.4.2.3. 0.96’ OLED zaslon

OLED zaslon služi za ispisivanje zadnje zabilježenih vrijednosti i informacije prilikom paljenja uređaja. Svaki puta kada se uređaj upali, spaja se na WiFi, spaja se na MQTT poslužitelj (pretplaćuje se na teme) i počinje mjeriti parametar. Kada se spoji na WiFi ispisuje se IP adresa koju je uređaj dobio, ukoliko se ne uspije spojiti na WiFi ispisuje se poruka greške. Ukoliko se spojio na MQTT poslužitelj ispisuje se poruka uspješnosti, obrnuto ukoliko se nije

uspio spojiti ispisuje se poruka greške. Nakon spajanja mjere se vrijednosti te se ispisuje temperature, vlaga zraka i stanje svijetla (upaljeno, ugašeno).

U ovom radu se koristi 0x78 ekran. 0x78 predstavlja adresu I2C standarda, ova adresa je bitna samo kod pisanja koda te ne utječe na sam rad zaslona. Zaslon pokreće SSD1315 čip, koji je novija verzija vrlo popularnog SSD1306 čipa.



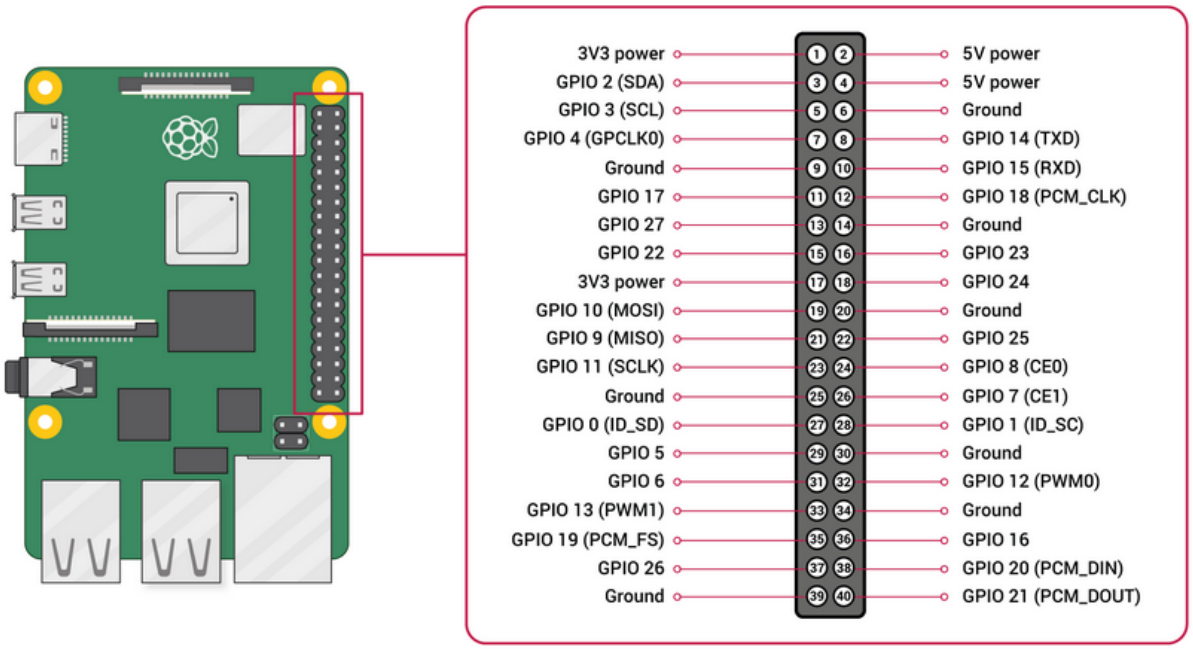
Slika 7: OLED SSD1315 pinout (Izvor: autor, 2023)

2.4.3. Raspberry Pi 3 B+

Raspberry Pi je računalo malih dimenzija no vrlo sposobnih mogućnosti. Ponaša se kao bilo koje drugo računalo koje pokreće operacijski sustav Linux, no uz to ima vrlo bogat sustav koji omogućuje spajanje raznih perifernih uređaja. Zbog toga je to savršeno računalo za ovaj rad. Službena stranica Raspberry Pi Foundation opisuje Raspberry Pi kao:

> „Raspberry Pi je jeftino računalo veličine kreditne kartice koje se priključuje na računalni monitor ili TV, a koristi standardnu tipkovnicu i miš. To je sposoban mali uređaj koji ljudima svih dobi omogućuje istraživanje računalstva i učenje programiranja u jezicima kao što su Scratch i Python. Sposoban je učiniti sve što biste očekivali od stolnog računala, od pregledavanja interneta i reprodukcije videa visoke razlučivosti do izrade proračunskih tablica, obrade teksta i igranja igrica. Štoviše, Raspberry Pi ima mogućnost interakcije s vanjskim svijetom i korišten je u širokom nizu projekata za izradu digitalnih proizvoda, od glazbenih strojeva i roditeljskih detektora do meteoroloških stanica i tweeting kućica za ptice s infracrvenim kamerama.“ („*What is a Raspberry Pi*?“, n.d.)

U ovom radu Raspberry Pi ima ulogu glavnog računala koje prima sve vrijednosti poslane preko MQTT protokola, pohranjuje te vrijednosti u lokalnu bazu podataka te komunicira sa .NET aplikacijom. Osim što šalje i prima podatke, zaduženo je i za paljenje/gašenje svijetla pomoću relay-a, koji će biti objašnjen kasnije.

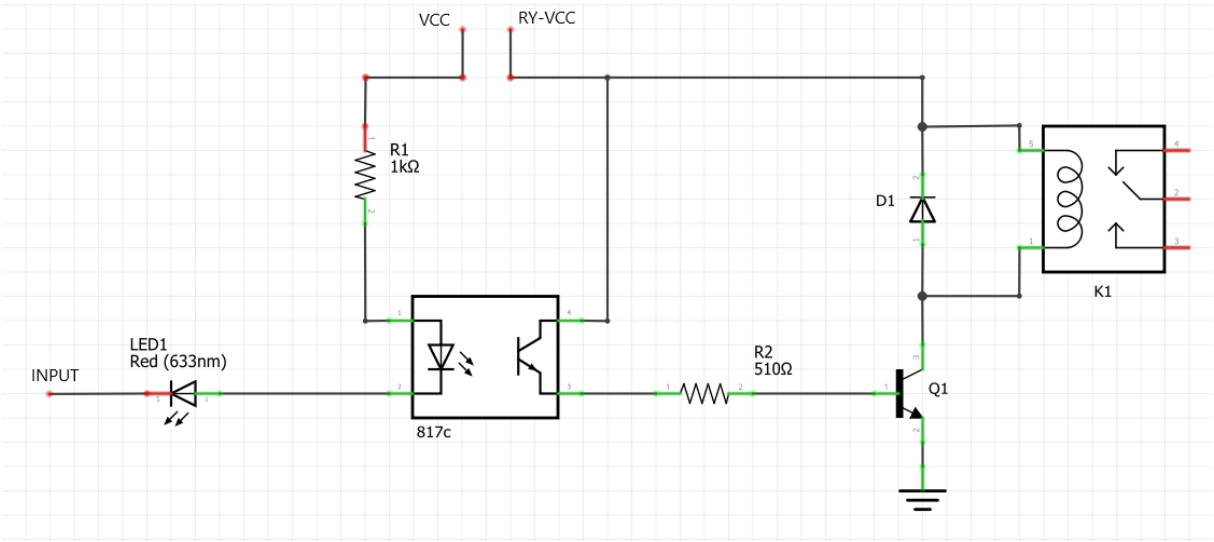


Slika 8: Raspberry Pi 3B+ pinout (Izvor: Raspberry Pi Foundation, n.d.)

2.4.3.1. Relay

Relay je elektronička komponenta koja služi za spajanje ili prekid strujnog kruga. Ovo je zapravo sklopka kojom se može upravljati preko digitalnog kanala za komunikaciju. U ovom projektu služi za paljenje/gašenje svijetla, te njime upravlja Raspberry Pi.

Karakteristike relay-a: 10A-250VAC, 10A-125VAC, 10A-30DC, 10A-28VDC.



Slika 9: 2 Relay Module shema (Izvor: autor, 2023)

3. Razrada teme

Cilj ovog rada je napraviti sustav pametne farme. Sustav mora, krajnjem korisniku, pružati uvid u trenutno stanje farme, povijesno stanje farme i kontrolu svjetla (paljenje/gašenje). Kako bi se omogućilo praćenje stanje farme, potrebni su senzori koji imaju mogućnost promatranja stanja, te umrežavanja, kako bi se omogućilo slanje skupljenih podataka. Skupljeni podaci se pohranjuju i prikazuju uživo. Kako bi se podaci pohranili, potrebno je računalo koje će ih primati. Najčešće se za ovaj problem, u IoT-u, koristi cloud. Pošto je implementacija rješenja pomoću cloud dosta skupa, a ovaj sustav će se koristiti u stvarnom svijetu, odabrana je opcija lokalnog spremanja podataka. Lokalno spremanje podataka riješeno je upotrebom Raspberry Pi računala. Ovdje se javlja još jedan problem, a to je komunikacija između krajnjeg uređaja sa prikupljanje podataka sa farme i računala koje će te podatke spremiti. Nakon provedenog istraživanja odlučeno je koristiti MQTT protokol. MQTT je vrlo efikasan protokol napravljen za upotrebu u IoT sustavima, stoga je to bio najlogičniji izbor za ovaj rad. Na kraju je potrebno sve te podatke prikazati krajnjem korisniku, zbog čega je izrađena .NET aplikacija.

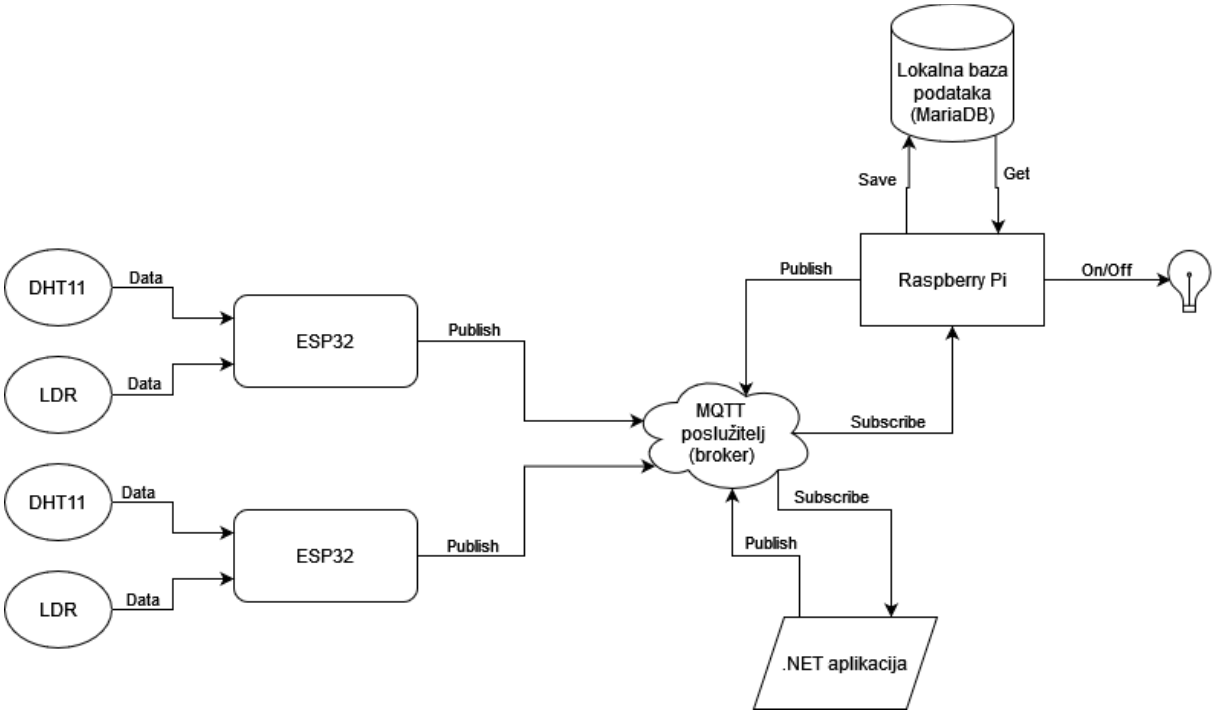
Ovo poglavlje se sastoji od detaljnog obrazloženja strukture sustava, programskog koda koji se nalazi na ESP32 uređajima, programskog koda koji se nalazi na Raspberry Pi računalu, baze podatak u koju se spremaju prikupljeni podaci i programskog koda .NET aplikacije.

3.1. Struktura sustava

Sustav se sastoji od 4 glavne komponente: ESP32 (uređaji koji prate stanje farme), Raspberry Pi (računalo koje prima, pohranjuje i šalje podatke), .NET aplikacija (grafičko korisničko sučelje) i MQTT poslužitelj (komunikacija između svih komponenti).

Kako bi se najlakše shvatila struktura sustava, ovdje će biti objašnjen tok podataka unutar sustava. ESP32 se nalazi unutar same farme, te prati svoju okolinu na temelju podataka koje prima sa 2 senzora (DHT11 i LDR). Mjerenja uzima svakih 5 minuta, prilikom kojeg šalje izmjerene podatke na MQTT poslužitelj. Podaci sa MQTT poslužitelja dolaze na Raspberry Pi. Raspberry Pi prima podatke sa tema na koje je pretplaćen. Na temelju unutarnje logike razlikuje vrstu podataka i sa kojeg senzora su stigli, te ih pohranjuje u odgovarajuću tablicu unutar baze podataka (MariaDB). Kada se prvi puta pokrene .NET aplikacija, ona šalje specijalnu poruku na MQTT temu, na temelju koje Raspberry Pi vraća zadnje zaprimljene podatke i podatke spremljene u bazi podataka. Zadnje zabilježeni podaci se vraćaju zbog toga

kako bi korisnik vidio stanje svoje farme uživo (pošto ESP32 šalje podatke svakih 5 minuta). Nakon ovih inicijalnih podataka, .NET više ne dobiva podatke sa Raspberry Pi računala, već podatke dobiva sa teme na koju ih šalju ESP32 uređaji. Raspberry Pi računalo je zaduženo za paljenje i gašenje svjetla. Zbog toga .NET aplikacija šalje poruku preko MQTT teme za paljenje svjetla. Korisnik prilikom paljenja svjetla ne može znati da li je svjetlo upaljeno odmah, već mora čekati do 5 minuta kada ESP32 pošalje podatke. Dijagram strukture sustava prikazan je na slici 11.

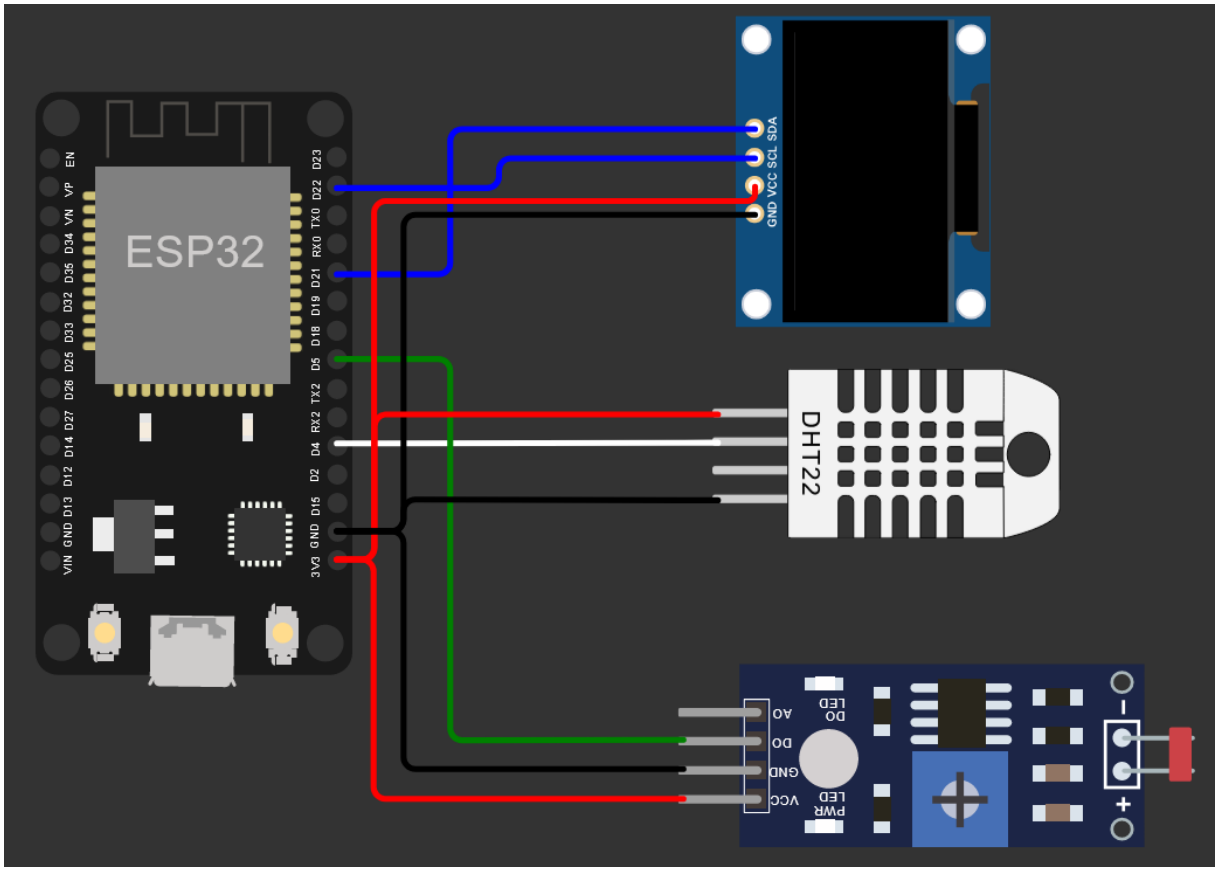


Slika 10: Struktura sustava (Izvor: autor, 2023)

Iz objašnjenja strukture se može vidjeti da se kompletna komunikacija odvija preko MQTT protokola. Razlog ovoga je taj da sustav može raditi neovisno o tome gdje se nalaze pojedine komponente. Zbog toga Raspberry Pi ne mora biti na samoj farmi, već može biti na nekoj drugoj lokaciji. Ovo može biti korisno kada korisnik mora, iz nekog razloga, pregledati Raspberry Pi i dijagnosticirati sustav. No unatoč tome što MQTT donosi ovu veliku prednost sustava, on predstavlja i najveću manu, zbog toga što je točka koja može srušiti cijeli sustav. Ukoliko prestane raditi MQTT poslužitelj gubi se jedini način komunikacije između sustava, čime se efektivno gubi korist sustava. Osim ovog katastrofalnog scenarija, mogu se dogoditi razni problem na raznim dijelovima mreže. Stoga poruke mogu kasniti ili uopće ne doći do krajnje komponente.

3.2. ESP32 i periferija

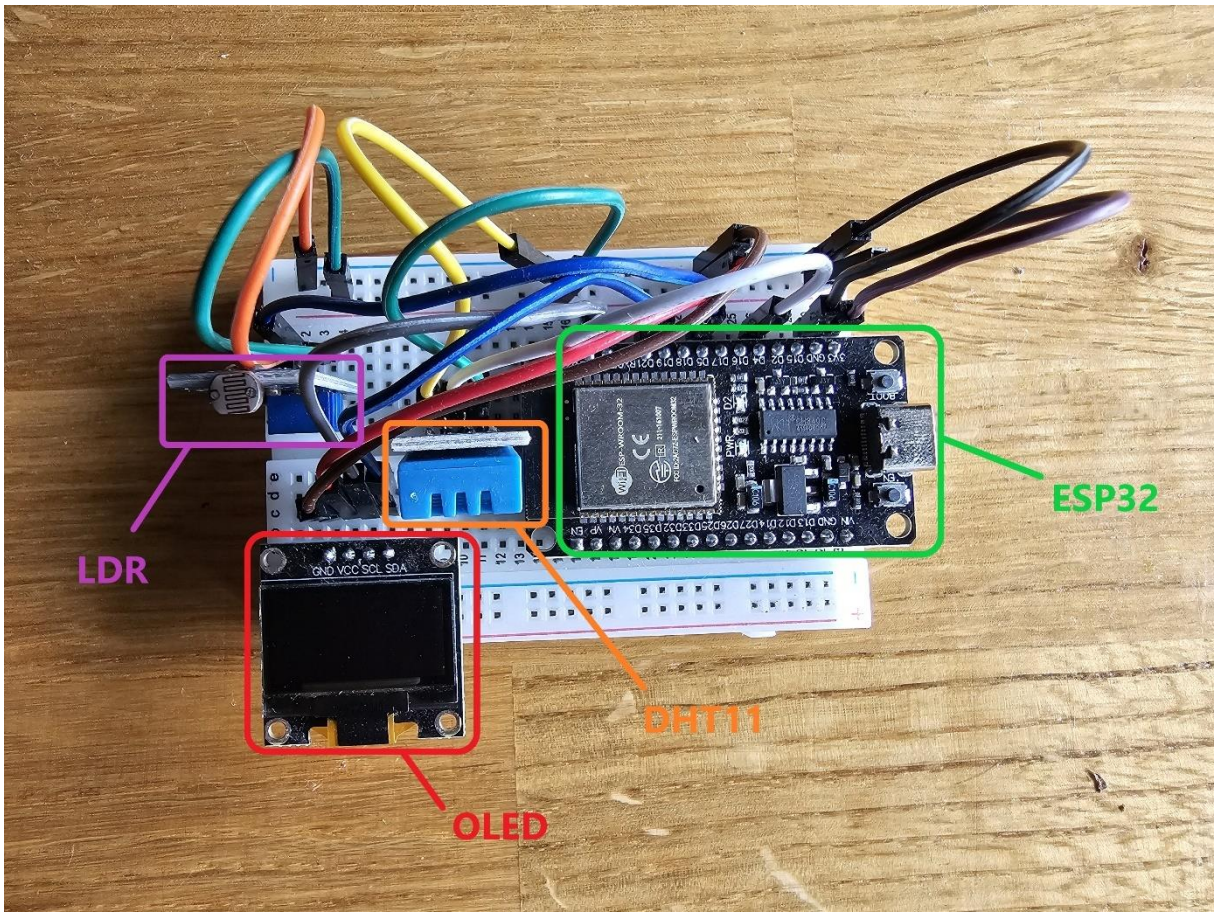
U ovom djelu rada biti će prikazan i objašnjen način spajanja senzora i zaslona na ESP32 uređaj. Sve će biti prikazano shematski te slikom uređaja u stvarnom svijetu.



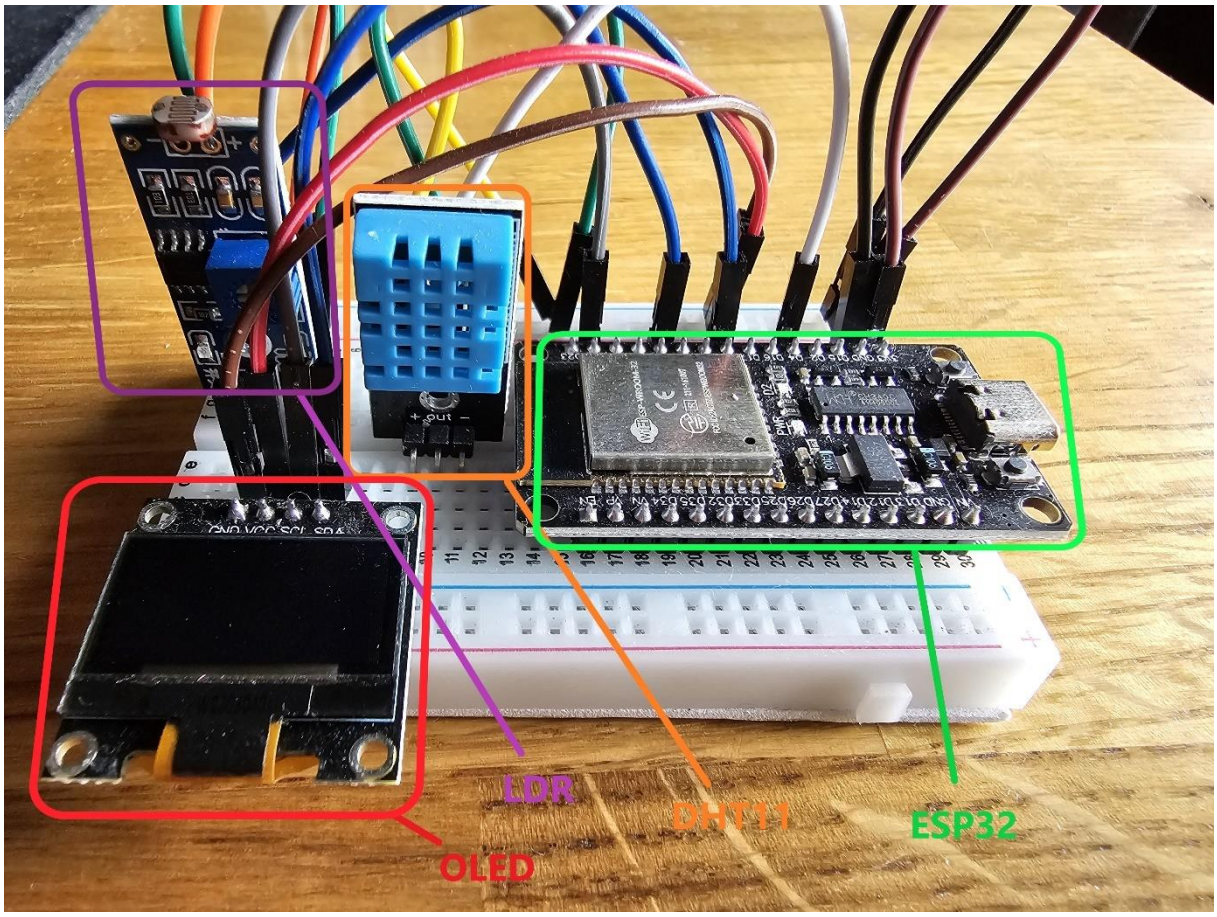
Slika 11: Shematski prikaz ESP32 i perifernih uređaja (Izvor: autor, 2023)

Na slici iznad (slika 12) je prikazana shema ESP32 i perifernih uređaja. Ovo je pojednostavljeni prikaz stvarnog uređaja. Crne žice predstavljaju uzemljenje (-), crvene žice predstavljaju napon (+), zelena žica služi za spajanje LDR senzora, bijela žica služi za spajanje DHT senzora, te plave žice služe za spajanje OLED zaslona. LDR senzor je spojen preko svojeg D0 pina na D5 pin na ESP32. DHT senzor je preko out pina spojen na D4 pin na ESP32. OLED zaslon koristi dva pina, SCL i SDA, SCl je spojena na D22, dok je SDA spojen na D21 pin na ESP32. Na slici je, umjesto DHT11 uređaja, prikazan DHT22 koji ima jedan pin više, te je prikazan LDR senzor koji također ima jedan pin viška. Ovo je napravljeno zbog nemogućnosti pronalaženja ispravnih komponenti unutar softwer-a koji je korišten za izradu sheme. Ovo ne predstavlja veliki problem, zbog toga što senzori prikazani na slici rade na isti način kao i oni koji su korišteni u ovom radu, jedina razlika je dodatna mogućnost koja se može koristiti upotrebom dodatnog pina.

Pošto je cilj da ovaj uređaj bude što manji, sve komponente su spojene na jednoj maloj testnoj pločici. Zbog toga u stvarnosti ovo izgleda „prenatrpano“, te se teško orijentirani po žicama koje spajaju sve komponente. Fotografija svih spojenih komponenti može se vidjeti na slikama 13 i 14.



Slika 12: Fotografija ESP32 i perifernih uređaja (Izvor: autor, 2023)



Slika 13: Fotografija ESP32 i perifernih uređaja iz drugog kuta (Izvor: autor, 2023)

Nigdje nije prikazan izvor napajanja. Programski kod, koji se izvodi na ESP32, je napisan tako da se uštedi čim više električne energije, zbog toga što je sustav zamišljen tako da radi pomoću baterije. Pošto WROOM-32 kit dolazi sa podrškom za bateriju i podrškom za napajanje preko USB-C porta, ovaj sustav radi jednako nebitno o načinu napajanja.

3.2.1. Programski kod

ESP32 je uređaj koji se nalazi unutar farme, te je zadužen za mjerenje parametara i slanje izmjerenih parametara na MQTT poslužitelj. Parametre mjeri pomoću dva senzora: DHT11 i LDR. DHT11 je senzor koji ima mogućnost mjerenja temperature i vlage zraka. LDR je fotosenzitivan senzor koji mjeri intenzitet svjetlosti. Pošto je ovaj uređaj napajan baterijom potrebno je uštedjeti što više energije. Zbog toga uređaj mjeri i šalje podatke u intervalima od 5 minuta, te između mjerenja ulazi u duboki san. U nastavku će biti objašnjen programski kod koji se izvodi na ovom uređaju.

```
#include <Arduino.h>
#include <U8g2lib.h>
#include <WiFi.h>
#include <string.h>
#include <DHT.h>
#include <stdlib.h>
#include <SPI.h>
#include <Wire.h>
#include <esp_wifi.h>
#include <PubSubClient.h>
#include <unistd.h>
```

Ovaj dio programskog koda odnosi se na definiranje biblioteka koje se koriste u program. Neke od važnijih su: U8G2lib.h, DHT.h, SPI.h, Wire.h i PubSubClient.h. U8G2lib.h je biblioteka koja je korištena kod ispisa na zaslonu. Uz nju se koriste Wire.h i SPI.h, koje omogućuju korištenje I2C i SPI uređaja sa ESP32 kontrolerom. DHT.h je biblioteka koja služi za komunikaciju DHT senzora sa ESP32. PubSubClient.h je biblioteka kojom se omogućuje rad sa MQTT protokolom

```
//definiranje ekrana
#ifdef U8X8_HAVE_HW_SPI
#endif
#ifdef U8X8_HAVE_HW_I2C
#endif
U8G2_SSD1306_128X64_NONAME_F_HW_I2C u8g2(U8G2_R0, /* clock=*/ SCL, /*
data=*/ SDA, /* reset=*/ U8X8_PIN_NONE);  // High speed I2C
```

Ovaj dio koda se odnosi na definiranje parametara koji omogućuju korištenje OLED zaslona sa ESP32. Nakon definiranje parametara, kreiran je objekt tipa **U8G2_SSD1306_128X64_NONAME_F_HW_I2C koji je nazvan u8g2**. Objekt koji je kreiran ovim načinom koristi brzi I2C.

```
//definiranje senzora temperature i vlage (DHT11)
```

```
#define DHTpin 4
#define DHTtype DHT11

DHT dht(DHTpin, DHTtype);
```

Ovim kodom je definiran pin na koji je spojen DHT11 senzor i definiran je tip DHT senzora (DHT11). Nakon toga kreiran je objekt tipa DHT koji je nazvan dht.

```
//definiranje senzora svijetlosti
#define lightSensor 5
```

Kod iznad se odnosi na definiranje pina na koji je spojen LDR senzor.

```
//WiFi
const char* ssid = "#######";
const char* password = "#######";

WiFiClient espClient;
```

Kako bi se ESP32 spojio na WiFi potrebno je inicijalizirati varijable **ssid** i **password**, te u njih pohraniti naziv mreže i lozinku. U ovom kodu su pravi naziv i lozinka mreže zamijenjeni znakovima # kako bi se zaštitila privatnost. Nakon toga je kreiran objekt tima **WiFiClient** koji je nazvan **espClient**.

```
//MQTT
const char* mqtt_server = "test.mosquitto.org";
const char* mqtt_password = "#######";
const char* mqtt_ssid = "#######";
const char* mqtt_topic_temperatura = "SmartFarmKarlo/Temperatura1";
const char* mqtt_topic_vlaga = "SmartFarmKarlo/Vlaga1";
const char* mqtt_topic_svijetlo = "SmartFarmKarlo/Svijetlo1";
const int mqtt_port = 1883;

PubSubClient client(espClient);
```

Ovaj dio koda se odnosi na inicijalizaciju varijabli potrebnih za korištenje MQTT protokola. Varijabla **mqtt_server** predstavlja adresu MQTT poslužitelja (u ovom radu se koristi besplatni poslužitelj Mosquitto). U varijable **mqtt_password** i **mqtt_ssid** su pohranjene lozinka i id uređaja koji se spaja na poslužitelj. Varijable **mqtt_topic** predstavljaju nazive tema na koje uređaj šalje podatke, svaka tema ima svoj naziv, te svaki uređaj ima više različitih tema. Zadnja varijabla je **mqtt_port** u kojoj je pohranjen port poslužitelja. Na kraju je deklariran objekt tipa **PubSubClient** koji je nazvan **client**, te koji prima vrijednost prethodno deklariranog objekta **espClient** koji je tipa **WifiClient**.

```
void ConnectToWiFi(const char* ssid, const char* password){
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    int timeout = 0;

    while (WiFi.status() != WL_CONNECTED)
    {
        u8g2.clearBuffer();
        u8g2.drawStr(0, 10, "Connecting to WiFi...");
        u8g2.sendBuffer();
        timeout++;

        delay(100);
        if(timeout >= 100){
            u8g2.clearBuffer();
            u8g2.clearDisplay();
            u8g2.drawStr(0, 10, "Can't connect to WiFi");
            u8g2.drawStr(0, 25, "Restart in 10s");
            u8g2.sendBuffer();
            //deep sleep, timer je u mikrosekundama
            esp_sleep_enable_timer_wakeup(10000000);
            esp_deep_sleep_start();
```

```
        }
    }

    Serial.println(WiFi.localIP());

    u8g2.clearBuffer();
    u8g2.clearDisplay();
    u8g2.drawStr(0, 10, "Connected");
    u8g2.drawStr(0, 20, "IP:");
    u8g2.drawStr(20, 20, WiFi.localIP().toString().c_str());
    u8g2.sendBuffer();
    delay(2000);
}
```

Iznad je definirana funkcija nazvana **ConnectToWifi**. Ova funkcija prima parameter **ssid** i **password**, te je tipa void što znači da ne vraća vrijednost. Svrha ove funkcije je da spoji ESP32 na WiFi. Na početku funkcije je definiran način korištenja WiFi mreže kao **WIFI_STA**, što znači da je uređaj definiran kao “station” tj. može se samo spojiti na WiFi. Nako toga je pokrenut proces spajanja na WiFi pomoću **WiFi.begin**. Varijabla **timeout** je inicjalizirana kako bi se definirao broj pokušaja spajanja na WiFi. Petlja While služi za ispitivanje da li je uređaj spojen na WiFi, tu dolazi u igru varijabla **timeout** tako da se varijabla povećava svaki puta kada se ispita da li je uređaj spojen na WiFi. Tijekom ovog procesa na zaslon se ispisuje poruka da je uređaj u fazi spajanja na WiFi. Ukoliko se uređaj bezuspješno pokuša spojiti na WiFi 100 puta, na zaslonu se ispisuje poruka o neuspješnom spavanju te uređaj ulazi u duboki san na 10 sekundi, nakon čega se ponovno pokušava spojiti. Maksimalan duljina procesa pokušaja spajanja je 10 sekundi zbog delay-a od 100 milisekundi, kada se 100 pokušaja spavanja pomnoži sa 100 milisekundi delay-a dobije se 10,000 milisekundi tj. 10 sekundi. Ukoliko se uređaj uspješno spoji na WiFi ispisuje se poruka na zaslonu i u konzoli (ukoliko se konzola motri preko vanjskog uređaja), unutar poruke se ispisuje i IP adresa uređaja. Na kraju je stavljen delay od 2 sekunde kako bi korisnik mogao vidjeti poruku.

```
void ConnectToMqtt(const char* mqttServer, int mqttPort){
    client.setServer(mqttServer, mqttPort);
```

```
    while (!client.connected())
    {
        u8g2.clearBuffer();
        u8g2.clearDisplay();
        u8g2.drawStr(0, 10, "Connecting to MQTT");
        u8g2.drawStr(0, 25, "broker...");
        u8g2.sendBuffer();

        delay(2000);

        client.setServer(mqtt_server, mqtt_port);
        client.connect("ESP32_client");
    }
    u8g2.clearBuffer();
    u8g2.clearDisplay();
    u8g2.drawStr(0, 10, "Broker connected");
    u8g2.sendBuffer();
    delay(2000);
}
```

Iznad je definirana funkcija nazvana **ConnectToMqtt**. Funkcija služi za poevzivanje uređaja na MQTT poslužitelj. Prima dva parametra, prvi je adresa MQTT poslužitelja (brokera), a drugi je port MQTT poslužitelja koji se koristi. Na početku funkcije se poziva metoda **setServer** koja postavlja parametre potrebne za povezivanje na broker. **While** petlja osigurava da će se povezivanje vršiti tako dugo dok ono nije uspješno. Unutar while petlje se ispisuje poruka na zaslonu koja govori korisniku da je uređaj u procesu spajanja na broker. Ukoliko se jednom izvršavanju petlje uređaj ne poveže na broker, potrebno je opet zadati parametre za povezivanje, te nakon toga opet pokušati povezati uređaj. Nakon **while** petlje, izvršava se kod koji ispisuje na zaslon da je povezivanje sa brokerom uspješno. Ukoliko program izađe iz **while** petlje znači da je povezivanje uspješno, zbog toga što će se uređaj pokušavati spojiti tako dugo dok spajanje nije uspješno.

```
void setup(void) {
```

```
    Serial.begin(115200);
    u8g2.begin();
    dht.begin();
    u8g2.setFont(u8g2_font_t0_12b_tf);
    pinMode(lightSensor, INPUT);
    delay(1000);

    ConnectToWiFi(ssid, password);
    ConnectToMqtt(mqtt_server, mqtt_port);

    u8g2.clearBuffer();
    u8g2.clearDisplay();
    u8g2.drawStr(0, 10, "Measuring...");
    u8g2.sendBuffer();
}
```

Setup je preddefinirana funkcija koja se izvršava prilikom prvog pokretanja programa, te ona služi za prvobitnu inicijalizaciju svih ostalih funkcija. **Serial.begin()** je naredba kojom se postavlja brzina prijenosa podataka u bitovima po sekundi, u ovom radu se koristi vrijednost od 115200 zbog toga što je onda definirana od strane proizvođača ESP32. Nakon toga se "pokreću" senzori sa **begin()** metodama, LDR senzor se pokreće drugačije na način da se definira pin na koji je on spojen kao **INPUT**. Naredba **u8g2.setFont()** poziva metodu koja postavlja font koji se koristi kod ispisa na zaslon. Nakon pokretanja senzora pozivaju funkcije koje su prije objašnjenje (**ConnctToWiFi()** i **ConnectToMqtt()**), te se njima uređaj prilikom paljenje povezuje na WiFi i broker. Na kraju se čisti sve što je ispisano na zaslonu i u njegovom buffer, te se ispisuje poruka kojom korisnik zna da uređaj trenutno mjeri parametar farme.

```
char tempBuffer[4];
char humBuffer[5];
int light;
char lightChar[6];
```

Ovo su varijable koje se koriste u glavnoj funkciji programa. **tempBuffer** je polje tipa **char** koje služi za pohranu vrijednosti temperature koja se dobiva sa DHT11 senzora, **humBuffer** je

identična varijabla drugog imena i koja služi za pohranu vrijednosti vlage. Varijabla **light** je tipa **int** i ona služi za pohranu vrijednosti dobivene sa LDR senzora. Pošto je LDR digitalni senzor on vraća samo vrijednost 0 ili 1, zbog toga je tu vrijednost potrebno pretvoriti u char. Radi toga je deklarirano polje **lightChar**.

```
void loop(void) {

    delay(500);

    //citanje vrijednosti temperature i vlage
    //dtostrf() pretvara iz float u char
    dtostrf(dht.readTemperature(), 4, 1, tempBuffer);
    dtostrf(dht.readHumidity(), 4, 1, humBuffer);

    //citanje vrijednosti svijetlosti, 1 ugaseno, 0 upaljeno
    //sprintf() pretvara int u char
    light = digitalRead(lightSensor);
    sprintf(lightChar, "%d", light);

    //ova vrijednost delya najbolje radi zbog vremena potrebnog za čitanje
sa senzora
    delay(300);

    u8g2.clearBuffer();
    u8g2.clearDisplay();
    u8g2.drawStr(0, 10, "Temperatura:");
    u8g2.drawStr(76, 10, tempBuffer);
    u8g2.drawStr(102, 10, "C");
    u8g2.drawStr(0, 25, "Vlaga:");
    u8g2.drawStr(40, 25, humBuffer);
    u8g2.drawStr(66, 25, "%");
```

```
    if(light)
        u8g2.drawStr(0, 40, "Svijetlo je ugaseno");
    else
        u8g2.drawStr(0, 40, "Svijetlo je upaljeno");

    u8g2.sendBuffer();

    client.publish(mqtt_topic_temperatura, tempBuffer);
    delay(1000);
    client.publish(mqtt_topic_vlaga, humBuffer);
    delay(1000);
    client.publish(mqtt_topic_svijetlo, lightChar);
    delay(1000);

    //min * microsec = koliko minute ce spavati
    unsigned int microsec = 60000000;
    esp_sleep_enable_timer_wakeup(5 * microsec);
    esp_deep_sleep_start();
}
```

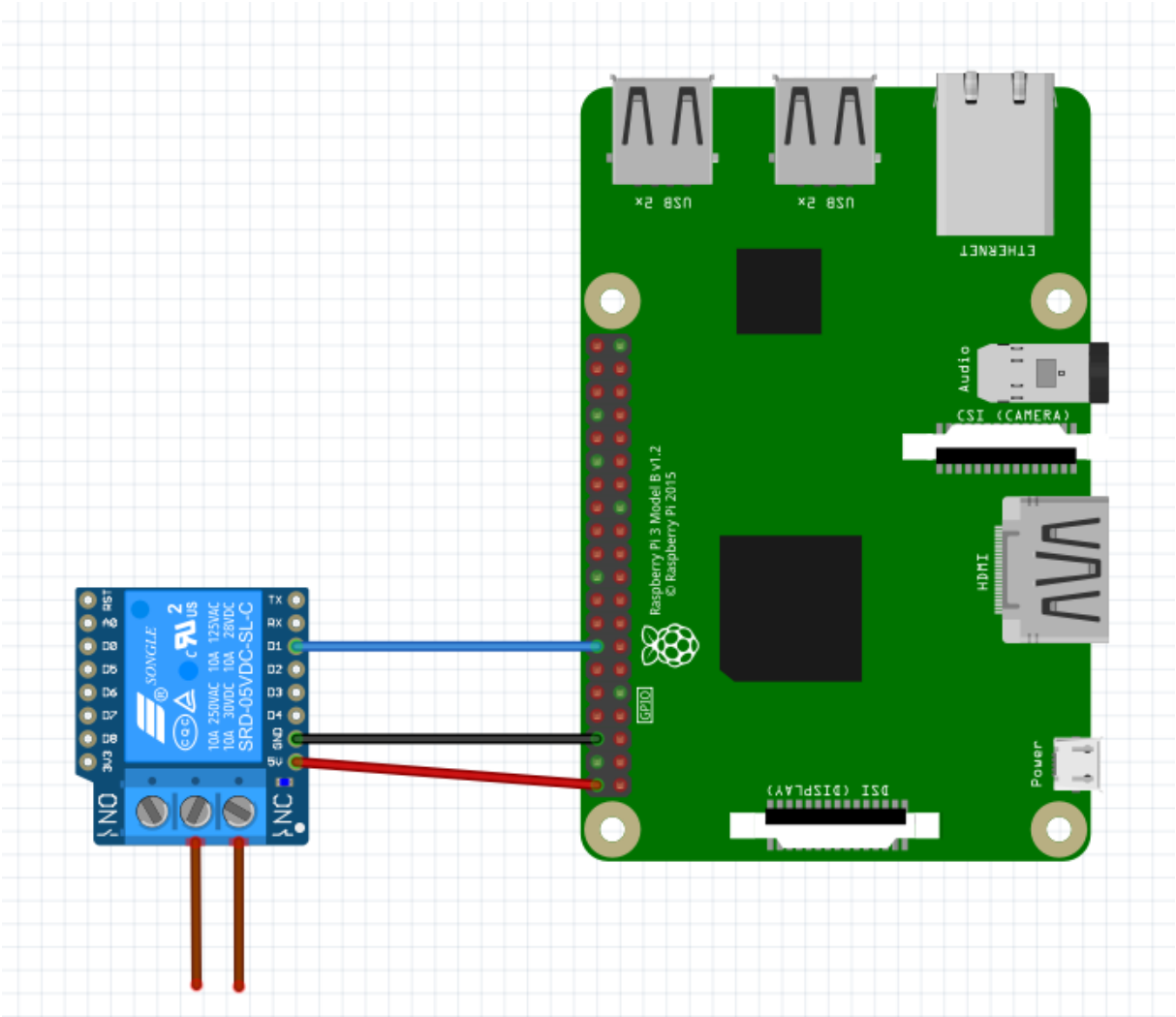
Funkcija **loop()** je preddefinirana funkcija unutar Arduino.h biblioteke, a govori uređaju da se konstantno izvršava nakon **setup()** funkcije. Na početku funkcije je čitanje vrijednosti sa senzora. Vrijednosti sa DHT senzora se čitaju na temelju metoda **readTemperature()** i **readHumidity()**, te se one nalaze unutar metode **dtostrf()** koja pretvara tip **double** u tip **char**. Na ovaj način je moguće spremiti vrijednost senzora u polje tipa **char**. Nakon čitanja sa DHT senzora, čita se sa LDR senzora preko metode **digitalRead()**, nakon čega se poziva metoda **sprint** koja pretvara vrijednost tipa **int** u tip **char**, kako bi se vrijednost spremila u polje tipa **char**. Vrijednosti je potrebno pretvoriti u **char** kako bi se mogle poslati preko MQTT protokola. Nakon koda za čitanje senzora stavljen je **delay** od 300 milisekundi, kroz testiranje je zaključeno da ovo vrijeme najbolje odgovara zbog toga što čitanje sa senzora može biti sporo. Nakon što su vrijednosti pročitane ispisuju se na zaslonu. Ispis na zaslonu moguć je pozivom metode **drawStr()** koja prima parametar za postavljanje pozicije kursora i tekst koji se treba ispisati. Pozicija kursora se postavlja zadavanjem x i y koordinata, radi kao i obični koordinatni

sustav gdje rezolucija ekrana predstavlja osi koordinatnog sustava. Za ispis stanja svijetla je potrebno dodatno pisanje koda kako bi se razlikovalo upaljeno i ugašeno stanje. Stoga je napisan kod koji ukoliko je vraćena vrijednost sa senzora jednaka 1, ispisuje se da je svjetlo ugašeno, suprotno ako je vrijednost 0 svijetlo je upaljeno (ovo nije greška, proizvođač senzora je dizajnirao senzor koji vraća 1 ako nema dovoljno svijetlosti i vraća 0 kada ima dovoljno svijetlosti). Metoda **drawStr()** sprema sve u jedan buffer koji se na kraju šalje na zaslon metodom **sendBuffer()** te se tekst ispisuje na zaslon. Nakon ispisa na zaslon potrebno je poslati dobivene vrijednosti na broker. Ovo je izvedeno pomoću metode **publish()**, koja prima parameter za naziv teme i vrijednost koja se šalje. Između svakog slanje poruke nalazi se delay od 1 sekunde, ovo je način na koji se osiguralo dolaženje poruka serijski s obzirom na kašnjenje u mreži. Na kraju se nalazi kod koji stavlja uređaj u stanje dubokog sna. U varijabli **microsec** je pohranjena vrijednost koja predstavlja koliko mikro sekunda se nalazi u jednoj minuti. Ovo je napravljeno zbog jednostavnosti izračuna vremena koje uređaj provodi u dubokom snu. Za definiranje vremena dubokog sna koristi se metoda **esp_sleep_enable_timer_wakeup()** koja prima vrijednost varijable **microsec** pomnoženom sa brojem minuta. Nakon definiranja vremena potrebno je pozvati metodu **esp_deep_sleep_start()** kojom uređaj ulazi u duboki san. Nakon što se uređaj probudi iz dubokog sna izvodi se funkcija **setup()**.

3.3. Raspberry Pi računalo

3.3.1. Raspberry Pi i periferija

Raspberry Pi je centralno računalo koje dohvaća, pohranjuje i šalje sve prikupljene podatke sa svih senzora, no ima još jedan zadatak, a to je paljenje i gašenje svijetla. Kako bi se omogućila ova funkcija potrebno je spojiti relay na Raspberry Pi. Detaljni prikaz spajanja relay-a na Raspberry Pi, prikazan je na slici 15.



Slika 14: Shematski prikaz Raspberry Pi računala i relay-a (Izvor: autor, 2023)

3.3.2. Programski kod

```
import paho.mqtt.client as mqtt
import paho.mqtt.publish as publish
from flask import Flask, render_template
import RPi.GPIO as GPIO
import time
from time import sleep
import mysql.connector as mariadb
from datetime import date
import datetime
```

Kod počinje sa deklaracijom biblioteka. Python ima drugačiji način korištenja biblioteka od C jezika. Na primjer, kreiranje objekta neke klase može se napraviti direktno kod deklariranja biblioteke u kojoj se ta klasa nalazi. Drugi primjer je to da se može pozvati samo određena klasa unutar biblioteke, no tada se ne može koristiti cijela biblioteka.

```
#Lights controll
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
relay_pin = 23
GPIO.setup(relay_pin,GPIO.OUT)
```

U ovom djelu koda je deklarirano sve što je potrebno za korištenja reley-a. Uz metodu **setmode()** postavlja se cijelo GPIO sučelje u BCM način rada. Nakon toga uklanjamo upozorenja koja bi se ispisivala prilikom svakog pokretanja programa (ovo nije nužno, no korišteno je radi boljeg izgleda programa, upozorenja se ispisuju samo radio korištenja BCM načina rada). Nakon toga se pin, na koji je spojen relay, postavlja kao output, tj. taj pin ne prima podatke već ih šalje prema van.

```
#MQTT
MQTT_SERVER = "test.mosquitto.org"
MQTT_TOPIC_TEMP = "SmartFarmKarlo/Temperatura"
MQTT_TOPIC_HUM = "SmartFarmKarlo/Vlaga"
```

```
MQTT_TOPIC_LIGHT = "SmartFarmKarlo/Svijetlo"
MQTT_TOPIC_SEND_DATA = "SmartFarmKarlo/DataSet"
```

Ovdje su deklariranje sve varijable koje služe za korištenje MQTT protokola. Prva varijabla predstavlja adresu brokera, dok ostale služe kao teme preko kojih se šalju i primaju poruke.

```
#Databse
mariadb_connection = mariadb.connect(user='##########',
    password='##########', database='##########', host='##########',
    port='##########')
cursor = mariadb_connection.cursor()
```

Kako bi program mogao raditi sa bazom podataka potrebno je definirati opcije spajanja i postaviti kursor. Opcije spajanja definiraju se pomoću metode **connect()** koja prima sljedeće parametre: naziv korisnika, lozinka za pristup, naziv baze podataka, adresa poslužitelja i broj porta poslužitelja. U ovom radu se koristi lokalna baza podataka stoga je adresa poslužitelja adresa samog uređaja (localhost). Nakon toga se vrijednost kursora dohvaća naredbom **mariadb_connection.cursor()**. Privatni podaci su zamijenjeni znakovima # kako bi se zaštitila privatnost.

```
today = date.today()
averageTemp = 0
averageHum = 0
messageCounterTopic1 = 0
firstBoot = True
tempMessage1 = "0"
humMessage1 = "0"
lightMessage1 = "Ugaseno"
```

Ove varijable se koriste za razne dijelove programa, te će biti objašnjene na dijelovima programa na kojima se koriste. Deklarirane su izvan funkcija kako bi se mogle koristiti unutar više funkcija.

```
def set_insert_id(table):
    sql_statement_count = 'SELECT COUNT(*) FROM ' + table
```

```
    cursor.execute(sql_statement_count)
    return    int(str(cursor.fetchall()).replace("[",    "").replace("]",
"").replace("(", "").replace(")", "").replace(",", "")) + 1


idSensor1 = set_insert_id("sensor1_data")
```

Funkcija **set_insert_id()** služi za dohvaćanje ID-a zadnjeg zapisa u tablici. Prima naziv tablice te vraća ID zadnjeg zapisa. Ova funkcija se zove samo prilikom prvog pokretanja programa kako bi u slučaju da tablica nije prazna sljedeći zapisi imali ispravan ID. Na kraju funkcije se vraća vrijednost ID-a naredbom **return**. Prilikom vraćanja potrebno je podatak pretvoriti u int, te je potrebno maknuti sve neželjene znakove, zbog toga što se iz baze dohvaćaju podaci u JSON formatu. Ispod funkcije je deklarirana varijabla u koju se pohranjuje vrijednost koju funkcija vraća.

```
def insert_into_database(table, ID, temperature, humidity, lights, today):
    sql_statement = 'INSERT INTO ' + table + ' (id, temperature, humidity,
     lights, log_date) VALUES (%s, %s, %s, %s, %s)';
    insert_data = (ID, temperature, humidity, lights, today)
    cursor.execute(sql_statement, insert_data)
    mariadb_connection.commit()
    print("Data saved")
    print("Table = " + table + " - id = " + str(ID) + " - temp = " +
     str(temperature) + " - hum = " + str(humidity) + " - lights = " +
     lights)
     print("**********************************************************
     ******************")
```

Funkcija **insert_into_database()** služi za pohranjivanje podataka unutar baze podataka. Prima parametre: naziv tablice, ID zapisa, vrijednost temperature, vrijednost vlage, vrijednost svijetlosti i vrijednost dana pohrane. Varijabla **sql_statement** sadrži standardnu sql naredbu koja služi za umetanje novog zapisa u tablicu. Varijabla **insert_data** sadrži vrijednosti koje će se zapisati u tablicu. Metodom **execute()** izvršava se navedena naredba, te se nakon toga metodom **commit()** spremaju vrijednosti u tablicu. Nakon izvršenja naredbe ispisuje se ime tablice te vrijednosti koje su pohranjene. Nakon svakog ispisa dodani su znakovi * kako bi se lakše vidjeli pojedini zapisi.

```
def read_from_database(table, topic):
    sql_statement = 'SELECT COUNT(*) FROM ' + table
    cursor.execute(sql_statement)
    count  =  int(str(cursor.fetchall()).replace("[",  "").replace("]",
     "").replace("(", "").replace(")", "").replace(",", ""))
    x = count - 60
    if(x <= 0):
        x = 1
    while x != count:
        sql_statement = 'Select temperature from ' + table + ' where (id='
            + str(count) + ')'
        cursor.execute(sql_statement)
        result  =  "          "  +  str(cursor.fetchall()).replace("[",
                   "").replace("]",  "").replace("(",  "").replace(")",
                   "").replace("'", "").replace(",", "") + " C    |    "

        sql_statement = 'Select humidity from ' + table + ' where (id=' +
            str(count) + ')'
        cursor.execute(sql_statement)
        result    =    result    +    str(cursor.fetchall()).replace("[",
                  "").replace("]",  "").replace("(",  "").replace(")",
                  "").replace("'", "").replace(",", "") + " %     |    "

        sql_statement = 'Select lights from ' + table + ' where (id=' +
            str(count) + ')'
        cursor.execute(sql_statement)
        result    =    result    +    str(cursor.fetchall()).replace("[",
                  "").replace("]",  "").replace("(",  "").replace(")",
                  "").replace("'", "").replace(",", "") + "    |    "

        sql_statement = 'Select log_date from ' + table + ' where (id=' +
            str(count) + ')'
        cursor.execute(sql_statement)
```

```
            result = result + str(cursor.fetchall()).replace("datetime.date",
                "").replace("[",      "").replace("]",      "").replace("(",
                        "").replace(")", "").replace("'", "").replace(",", "-
                        ").replace(" ", "")


            count -= 1
            publish.single(topic, result, hostname=MQTT_SERVER)
            time.sleep(0.1)
```

Funkcija **read_from_database()** služi za čitanje vrijednosti iz baze podataka i slanje istih na broker. Prima parametre: naziv tablice i naziv MQTT teme. Funkcija započinje izvršenjem sql naredbe koja vraća broj zapisa unutar tablice, te se ta vrijednost pohranjuje unutar varijable **count**. Ova varijabla, uz varijablu **x**, služi za dohvaćanje zadnjih 60 zapisa. Ovo je napravljeno kako korisnik ne bi vidio sve vrijednosti unutar baze podataka, već samo vrijednosti sa senzora u zadnjih 60 dana. Unatoč tome, sve vrijednosti se čuvaju u bazi podataka, za slučaj da korisnik želi detaljniji uvid. Kako bi se osiguralo da se čita samo 60 zadnjih zapisa, unutar varijable x pohranjena je vrijednost varijable **count** od koje je oduzeta vrijednost 60, ukoliko je varijabla **x** manja ili jednaka 0 tada ona postaje 1. Nakon toga se izvodi **while** petlja, unutar koje se u varijablu **result** pohranjuju sve vrijednosti iz stupca određenog retka. Redak za čitanje se određuje na temelju varijable **count**, koja predstavlja ID samog zapisa. Nakon svakog pročitanog retka, **count** se smanjuje za 1, te se čitanje nastavlja tako dugo dok **count** nije jednak **x**. Ovim načinom se podaci čitaju od posljednjeg do prvog (prvi može biti ili na poziciji 1 ili na poziciji **count** - 60). Nakon čitanja retka se šalju podaci naredbom **publish.single()**, nakon koje je stavljen **delay(0.1)**. Svrha ovog delay-a je osigurati da se poruke šalju sinkrono kako bi broker mogao razlikovati razne poruke.

```
def on_connect(client, userdata, flags, rc):
    print("Connection code:" +str(rc))

    #subscribing to first sensor
    client.subscribe(MQTT_TOPIC_TEMP + "1")
    client.subscribe(MQTT_TOPIC_HUM + "1")
    client.subscribe(MQTT_TOPIC_LIGHT + "1")
```

Funkcija **on_connect()** služi za pretplaćivanje programa na željene MQTT teme. Ovo je predefinirana funkcija **paho** biblioteke. Kada se pozove ispiše da li je povezivanje na broker bilo uspješno, te se pomoću metode **subscribe()** pretplaćuje na teme.

```
def on_message(client, userdata, msg):
    global tempMessage1, humMessage1, lightMessage1, today
    global averageTemp, averageHum, messageCounterTopic1, firstBoot

    if(msg.topic == "SmartFarmKarlo/Temperatura1"):

        if(str(msg.payload) == "b'FirstConnection'"):
            time.sleep(0.1)
            publish.single("SmartFarmKarlo/Temperatura1",      "~"      +
                    tempMessage1, hostname=MQTT_SERVER)
            time.sleep(0.1)
            publish.single("SmartFarmKarlo/Vlaga1",  "~"  +  humMessage1,
             hostname=MQTT_SERVER)
            time.sleep(0.1)
            publish.single("SmartFarmKarlo/Svijetlo1",        "~"       +
                    lightMessage1, hostname=MQTT_SERVER)
            read_from_database("sensor1_data",
                                "SmartFarmKarlo/DataSet1")

        else:
            if '~' not in str(msg.payload):
                print("*******" + msg.topic + "*******")
                tempMessage1       =       str(msg.payload).replace("'",
                           "").replace("b", "")
                print("Temperature: " + tempMessage1)
                averageTemp += float(tempMessage1)

    if(msg.topic == "SmartFarmKarlo/Vlaga1"):
        if '~' in str(msg.payload):
            return
```

```
        humMessage1 = str(msg.payload).replace("'", "").replace("b", "")
        print("Humidity: " + humMessage1)
        averageHum += float(humMessage1)

    if(msg.topic == "SmartFarmKarlo/Svijetlo1"):
        if '~' in str(msg.payload):
            return

        if(str(msg.payload) == "b'LightOn'"):
            GPIO.output (relay_pin,GPIO.LOW)
            print(str(msg.payload))
            return

        if(str(msg.payload) == "b'LightOff'"):
            GPIO.output (relay_pin,GPIO.HIGH)
            print(str(msg.payload))
            return

        messageCounterTopic1 += 1
        lightMessage1  =  str(msg.payload).replace("'",  "").replace("b",
            "")
        if(lightMessage1 == "0"):
            lightMessage1 = "Upaljeno"
            print("Lights: " + lightMessage1)
        else:
            lightMessage1 = "Ugaseno"
            print("Lights: " + lightMessage1)
        print(datetime.datetime.now())
        if today != date.today():
            global idSensor1
            idSensor1 += 1
```

```
averageTemp /= messageCounterTopic1

averageHum /= messageCounterTopic1

averageTemp = round(averageTemp, 2)

averageHum = round(averageHum, 2)

insert_into_database("sensor1_data" ,idSensor1, averageTemp,
 averageHum, lightMessage1, today)

today = date.today()

messageCounterTopic1 = 0

averageTemp = 0

averageHum = 0
```

Funkcija **on_message** je predefinirana funkcija **paho** biblioteke, te služi za definiranje događaja koji će se dogoditi prilikom dolaska poruke na pretplaćenu temu. Na početku su deklarirane globalne varijable koje služe za pohranu vrijednosti koje se spremaju u bazu podataka i drugih podataka koji će biti objašnjeni kasnije. Poruke se razlikuju na temelju tema sa kojih su došle, te na temelju samog sadržaja poruke. Vrijednost teme se može dohvatiti preko naredbe **msg.topic**, dok se sadržaj poruke može dohvatiti preko naredbe **msg.payload**, gdje je **msg** proslijeđen samoj funkciji. Sustav je napravljen tako da svaka vrijednost (temperatura, vlaga, svijetlost) ima svoju temu. Tema „SmartFarmKarlo/Temperatura1“ je osim za vrijednost temperature zaslužna i za povezivanje sa .NET aplikacijom. Stoga kada dođe posebna poruka sa .NET aplikacije, ona mora poslati sve podatke iz baze podataka i zadnje zabilježene podatke sa senzora. Pošto je na ESP32 napisan kod koji osigurava sinkrono slanje podataka, ovaj kod skuplja sve te vrijednosti te ih akumulira i sprema u bazu samo jednom dnevno. Da li je započeo drugi dan, provjerava se na način da se u varijablu today spremi vrijednost naredbe **date.today()**, zatim se ispituje da li je **today = date.today()**, ukoliko je ova tvrdnja netočna prosjek akumuliranih podataka se sprema u bazu podataka te **today** poprima vrijednost **date.today()**. Ovo se proispituje na temi za svijetlo zbog toga što ta tema zadnja dobiva poruku sa senzora. Ova tema služi i za paljenje/gašenje svijetla, tako da se promjeni vrijednost pina na koji je spojen relay, za to su zaslužne naredbe **GPIO.output(relay_pin, GPIO.HIGH)** (gašenje svijetla) i **GPIO.output(relay_pin, GPIO.LOW)** (paljenje svijetla).

```
client = mqtt.Client()

client.on_connect = on_connect

client.on_message = on_message
```

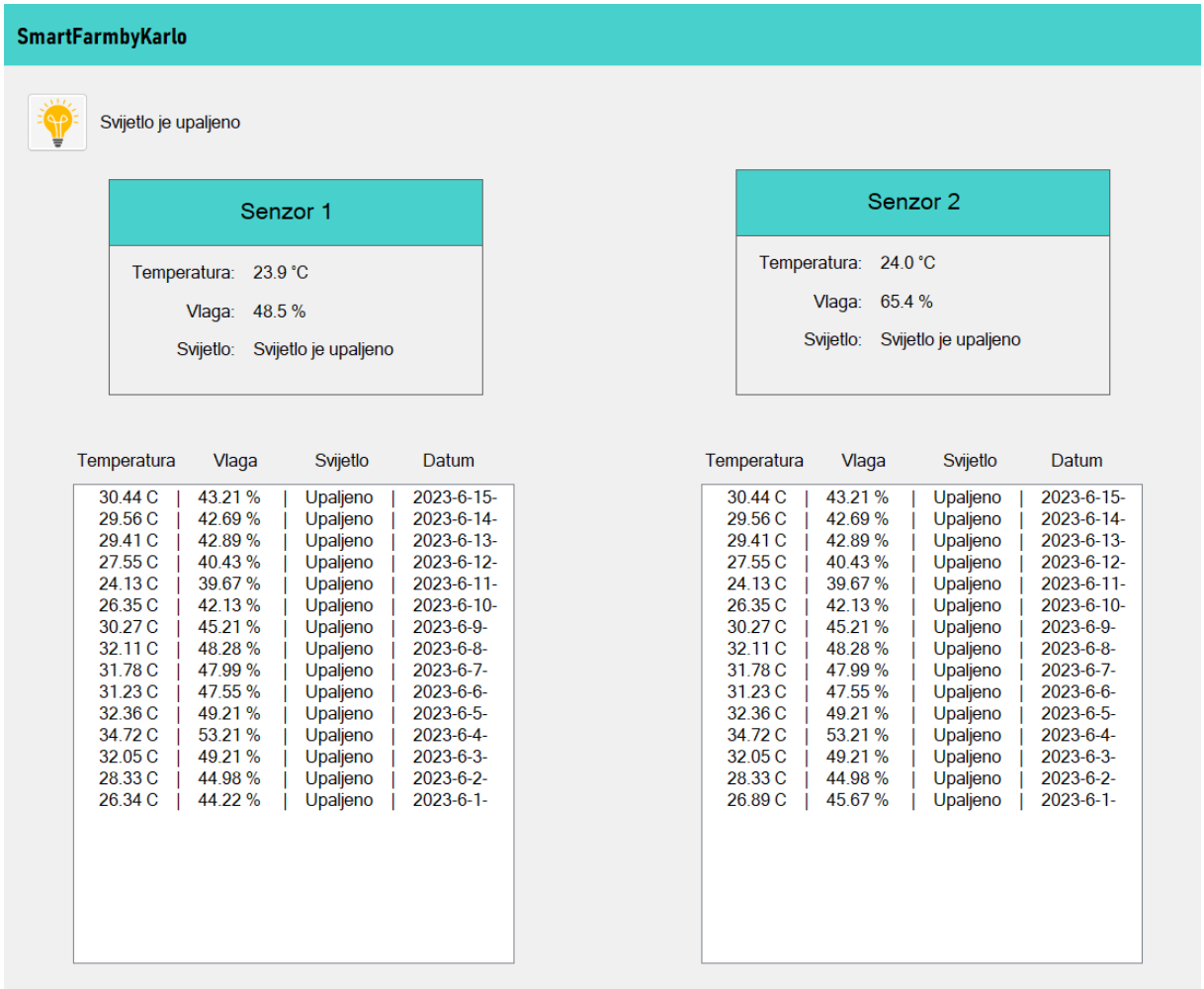
```
client.connect(MQTT_SERVER)


client.loop_forever()
```

Na kraju koda, izvan svih funkcija, nalaze se naredbe koje se izvršuju prilikom prvog pokretanja koda. Prvo se deklarira varijabla **client** koja je tipa **mqtt.Client()**. Nakon toga se definira koja funkcija se koristi prilikom spajanja na broker i prilikom dolaska poruke sa brokera. Nakon definiranja događaja spaja se na broker metodom **connect()** koja prima adresu brokera. Na kraju koda se naredbom **client.loop_forever()** osigurava da se program izvršava beskonačno mnogo puta.

3.4. .NET aplikacija

3.4.1. Dizajn

Dizajn aplikacije je vrlo jednostavan, te je napravljan u svrhu prikazivanja demo aplikacije. Sastoji se od 3 glavna djela: gumb za paljenje/gašenje svijetla, okvir za prikaz trenutnog stanja farme i okvir za prikaz povijesnog stanja farme. Gumb za paljenje/gašenje svijetla je jednostavan gumb sa dodanom pozadinom (ikona žarulje), pokraj kojeg se nalazi „label“ koji služi za ispis stanja svijetla. Ispod gumba se nalaze okviri kojima se prikazuje trenutno stanje farme, svi elementi unutar okvira su tipa „label“ te mijenjaju svoje stanje svaki puta kada stigne nova poruka sa senzora. Ispod svega se nalazi okvir koji je tipa „ListBox“, u njemu se ispisuju vrijednosti pohranjene u bazi podataka, radi tako da se prilikom stizanja nove poruke doda na dno liste. Prikaz izgleda aplikacije nalazi se na slici 16.



Slika 15: Dizajn .NET aplikacije (Izvor: autor, 2023)

3.4.2. Programski kod

```
using System;
using System.Text;
using System.Windows.Forms;
using uPLibrary.Networking.M2Mqtt.Messages;
using uPLibrary.Networking.M2Mqtt;
```

Kod započinje pozivanjem biblioteka. Prve tri biblioteke su osnovne biblioteke za razvoj .NET aplikacije. Dok su zadnje dvije biblioteke koje omogućuju rad sa MQTT protokolom.

```
//objekti senzora1
MqttClient sensor1Temp, sensor1Hum, sensor1Light, sensor1AllData, lightOnOff;

//objekti senzora2
MqttClient sensor2Temp, sensor2Hum, sensor2Light, sensor2AllData;

//Mqtt info
string broker = "test.mosquitto.org";
bool firstBoot = true;

char[] removedCharacters = {'~'};

private bool lightButtonClick = false;
private bool first = true;
```

U ovom djelu koda se kreiraju **MqttClient** objekti koji će kasnije služiti za povezivanje na broker. Osim toga se deklariraju varijable sa podacima za povezivanje na broker, te varijable koje se koriste za razne dijelove programa. Sve varijable biti će detaljnije objašnjene u nastavku.

```
public Form1() {
      InitializeComponent();
```

```
      //Topic senzora 1
      connectToBroker(broker,              "SmartFarmKarlo/Temperatura1",
"sensor1temp", sensor1Temp);
      connectToBroker(broker,   "SmartFarmKarlo/Vlaga1",   "sensor1hum",
sensor1Hum);
      connectToBroker(broker, "SmartFarmKarlo/Svijetlo1", "sensor1light",
sensor1Light);
      connectToBroker(broker,   "SmartFarmKarlo/DataSet1",   "dataSet1",
sensor1AllData);
      //connectToBroker(broker, "SmartFarmKarlo/Svijetlo1", "lightOnOff",
lightOnOff);

      //Topic senzora 2
      connectToBroker(broker,              "SmartFarmKarlo/Temperatura2",
"sensor2temp", sensor2Temp);
      connectToBroker(broker,   "SmartFarmKarlo/Vlaga2",   "sensor2hum",
sensor2Hum);
      connectToBroker(broker, "SmartFarmKarlo/Svijetlo2", "sensor2light",
sensor2Light);
      connectToBroker(broker,   "SmartFarmKarlo/DataSet2",   "dataSet2",
sensor2AllData);
    }
```

Funkcija Form1() je zadužena za pokretanje forme na kojoj se nalazi korisničko sučelje. Nakon inicijalizacije forme, poziva se funkcija koja služi za povezivanje programa na broker. Funkcija se poziva onoliko puta koliko ima tema, te se za svaki senzor program spaja na posebnu temu.

```
private void connectToBroker(string brokerAddress, string topic, string
ID, MqttClient client) {
      client = new MqttClient(brokerAddress);
      client.MqttMsgPublishReceived                                    +=
MqttClient_MqttMsgPublishReceivedTopic1;
      client.Subscribe(new string[] {
        topic
      }, new byte[] {
```

```
        MqttMsgBase.QOS_LEVEL_AT_LEAST_ONCE
      });
      client.Connect(ID);
      Console.WriteLine("Connected to " + topic);
      if (firstBoot) {
        client.Publish("SmartFarmKarlo/Temperatura1",
Encoding.UTF8.GetBytes("FirstConnection"),
MqttMsgBase.QOS_LEVEL_AT_MOST_ONCE, true);
        client.Publish("SmartFarmKarlo/Temperatura2",
Encoding.UTF8.GetBytes("FirstConnection"),
MqttMsgBase.QOS_LEVEL_AT_MOST_ONCE, true);
        firstBoot = false;
      }
    }
```

Funkcija **connectToBroker()** služi za povezivanje programa na broker. Prima parametre: adresa brokera, naziv teme, ID spajanja i objekt klijenta. Na temelju proslijeđenog objekta kreira se nova konekcija na broker. Kod kreiranje konekcije potrebno je definirati temu na koju se klijent pretplaćuje te kako će se slati poruke (**QOS_LEVEL_AT_LEAST_ONCE**). Nakon spajanja klijenta na broker, ispisuje se na koju temu je klijent spojen. Na kraju funkcije nalazi se **if** kojim se preispituje ukoliko je program prvi puta pokrenut. Ovo se preispituje na temelju prethodno deklarirane varijable **firstBoot**, koja se nakon ovog stavlja u **false**, kako bi se ovaj kod izveo samo jednom. Ovo je način na koji se dohvaćaju sve vrijednosti iz baze podataka, za svaki senzor.

```
private void MqttClient_MqttMsgPublishReceivedTopic1(object sender,
MqttMsgPublishEventArgs e) {
      var message = Encoding.UTF8.GetString(e.Message);

      //Poruke za senzor 1
      if (e.Topic == "SmartFarmKarlo/Temperatura1" && IsHandleCreated) {
        Console.WriteLine(message);
        string trimmedMessage = message.TrimStart(removedCharacters);
        sensorTemp1.Invoke((MethodInvoker)() => sensorTemp1.Text =
trimmedMessage + " °C"));
```

```
      }
      if (e.Topic == "SmartFarmKarlo/Vlaga1" && IsHandleCreated) {
        Console.WriteLine(message);
        string trimmedMessage = message.TrimStart(removedCharacters);
        sensorHum1.Invoke((MethodInvoker)(() => sensorHum1.Text =
trimmedMessage + " %"));
      }
      if (e.Topic == "SmartFarmKarlo/Svijetlo1" && IsHandleCreated) {
        Console.WriteLine(message);
        if (message == "1") {
          Console.WriteLine("Svijetlo je ugaseno");
          sensorLight1.Invoke((MethodInvoker)(() => sensorLight1.Text =
"Svijetlo je ugašeno"));
          if (first) {
            lightStatus.Invoke((MethodInvoker)(() => lightStatus.Text =
"Svijetlo je ugašeno"));
          }
        } else {
          Console.WriteLine("Svijetlo je upaljeno");
          sensorLight1.Invoke((MethodInvoker)(() => sensorLight1.Text =
"Svijetlo je upaljeno"));
          if (first) {
            lightStatus.Invoke((MethodInvoker)(() => lightStatus.Text =
"Svijetlo je upaljeno"));
          }
        }
      }

      if (e.Topic == "SmartFarmKarlo/DataSet1" && IsHandleCreated) {
        sensor1HistoryData.Invoke((MethodInvoker)(() =>
sensor1HistoryData.Items.Add(message)));
      }
```

```
      //Poruke za senzor 2
      if (e.Topic == "SmartFarmKarlo/Temperatura2" && IsHandleCreated) {
        Console.WriteLine(message);
        string trimmedMessage = message.TrimStart(removedCharacters);
        sensorTemp2.Invoke((MethodInvoker)(() => sensorTemp2.Text =
trimmedMessage + " °C"));
      }
      if (e.Topic == "SmartFarmKarlo/Vlaga2" && IsHandleCreated) {
        Console.WriteLine(message);
        string trimmedMessage = message.TrimStart(removedCharacters);
        sensorHum2.Invoke((MethodInvoker)(() => sensorHum2.Text =
trimmedMessage + " %"));
      }
      if (e.Topic == "SmartFarmKarlo/Svijetlo2" && IsHandleCreated) {
        Console.WriteLine(message);
        if (message == "1") {
          Console.WriteLine("Svijetlo je ugaseno");
          sensorLight2.Invoke((MethodInvoker)(() => sensorLight2.Text =
"Svijetlo je ugašeno"));
        } else {
          Console.WriteLine("Svijetlo je upaljeno");
          sensorLight2.Invoke((MethodInvoker)(() => sensorLight2.Text =
"Svijetlo je upaljeno"));
        }
      }

      if (e.Topic == "SmartFarmKarlo/DataSet2" && IsHandleCreated) {
        sensor2HistoryData.Invoke((MethodInvoker)(() =>
sensor2HistoryData.Items.Add(message)));
      }
    }
```

MqttClient_MqttMsgPublishReceivedTopic1 je predefinirana funkcija M2Mqtt biblioteke, te služi za definiranje događaja koji se izvodi prilikom dobivanja poruke. Na početku je deklarirana varijabla u koju se sprema vrijednost poruke, vrijednost poruke prvo treba biti enkodirana. Unutar funkcije je, uz pomoć **if** petlje, definirano što se događa prilikom primanje pojedine poruke. Za sve poruke, koje predstavljaju vrijednost senzora, je definirano da se ispišu na zadani **label** koji se nalazi na formi. Unutar obrađivanja poruke za svijetlo, potrebno je ispisati ispravnu tvrdnju. Zbog toga što na temu za svijetlo dolazi ili 0 ili 1, stoga je potrebno zamijeniti ovu vrijednost sa tekstom. Ovdje je još dodana funkcionalnost za paljenje i gašenje svijetla, na način da se kraj tog gumba ispiše ispravan tekst. Za drugi senzor je kod identičan, osim naziva tema, objekata i izostavljanja koda za paljenje/gašenje svijetla, zbog toga što on nije potreban.

```
private void LightControllbtn_Click(object sender, EventArgs e) {


      if (first) {


        lightOnOff = new MqttClient(broker);

        lightOnOff.MqttMsgPublishReceived                                    +=
MqttClient_MqttMsgPublishReceivedTopic1;

        lightOnOff.Subscribe(new string[] {

          "SmartFarmKarlo/Svijetlo1"

        }, new byte[] {

          MqttMsgBase.QOS_LEVEL_AT_LEAST_ONCE

        });

        lightOnOff.Connect("tewteaasdf");

        Console.WriteLine("Connected to light");

        first = false;

      }

      if (lightButtonClick) {

        lightOnOff.Publish("SmartFarmKarlo/Svijetlo1",
Encoding.UTF8.GetBytes("LightOn"),    MqttMsgBase.QOS_LEVEL_AT_MOST_ONCE,
true);

        Console.WriteLine("Svijetlo je upaljeno");

        lightButtonClick = false;
```

```
        lightStatus.Invoke((MethodInvoker)(() => lightStatus.Text =
"Svijetlo je upaljeno"));
      } else {
        lightOnOff.Publish("SmartFarmKarlo/Svijetlo1",
Encoding.UTF8.GetBytes("LightOff"), MqttMsgBase.QOS_LEVEL_AT_MOST_ONCE,
true);
        lightButtonClick = true;
        Console.WriteLine("Light turned off");
        lightStatus.Invoke((MethodInvoker)(() => lightStatus.Text =
"Svijetlo je ugaseno"));
      }
    }
```

Funkcija **LightControllbtn_Click()** služi za definiranje događaja koji će se izvesti prilikom pritiska na gumb. Na početku je iskorištena petlja **if** kako bi se odredilo da li je gumb prvi puta pritisnut. Ukoliko je gumb prvi puta pritisnut, tada je potrebno kreirati novi klijent, postupak je već prethodno objašnjen. Ispitivanje za prvi pritisak gumba je izvedeno na temelju prethodno deklarirane varijable **first**. Događaji koji će se dogoditi se razlikuju na temelju **if** i **else** petlje. Na početku se pokraj gumba ispisuje vrijednost trenutnog stanja svijetla (kod koji se nalazi unutar **MqttClient_MqttMsgPublishReceivedTopic1** funkcije). Za slučaj da se želi upaliti svijetlo: pritiskom na gumb šalje se poruka „LightOn“, te se mijenja sadržaj labele u poruku „Svijetlo je upaljeno“. Za slučaj da se želi ugasiti svijetlo postupak je isti samo sa drugačijom porukom i ispisom.

4. Zaključak

Svrha ovog rada bila je napraviti sustava koji će motriti trenutno stanje farme, pohranjivati zabilježene podatke, dati uvid korisniku u prošla i trenutna stanja farme, te dati korisniku kontrolu svijetla unutar farme.

Istraživanjem trenutnih trendova u IoT sustavima, napravljeno je rješenje koje je služilo kao putokaz kod izrade ovog rada. Ukratko sustav je zamišljen na sljedeći način. Unutar farme nalaze se više uređaja koji imaju senzore potrebne da bi pratili određene parametre farme, osim senzora imaju sposobnost umrežavanja, te sposobnost napajanja baterijom. Ovi uređaji služe samo za mjerenje podataka i slanje istih. Za svo obrađivanje podatak zaslužno je centralno računalo. Korisnik do podatak dolazi preko aplikacije na svom računalu. Sva komunikacija između komponenata je izvedena preko MQTT protokola.

Kroz izradu rada ostvareno je kompletno rješenje sustava koje je prethodno navedeno. Uređaji unutar farme su ESP32 kontroleri, na koje su spojeni senzori temperature, vlage zraka, i razine svjetlosti. ESP32 stavlja podatke na MQTT temu, sa koje ih preuzima Raspberry Pi računalo. Raspberry Pi je centralno računalo, koje prima, obrađuje, pohranjuje i šalje podatke. Podaci su pohranjeni lokalno, pomoću Mariadb baze podataka. Za korisnika je izrađena .NET aplikacija, koja se koristi pomoću Windows Forms-a. Korisnik kroza aplikaciju dobiva uvid u trenutno stanje svih senzora, prošla stanja svih senzora (do 60 dana) i kontrolu svijetla. Kao broker koristi se Mosquitto.

5. Popis literature

Artturi Jalli (2022). *What is C#?* Preuzeto 04.08.2023. s https://builtin.com/software-engineering-perspectives/c-sharp

Artturi Jalli (2022). *What is MariaDB?* Preuzeto 07.08.2023. s https://builtin.com/data-science/mariadb

Bilal [slika] (2022). *Interface OLED with ESP32 using ESP-IDF*. Preuzeto 07.08.2023 s https://esp32tutorials.com/oled-esp32-esp-idf-tutorial/

Chinmoy Lenka (2023). *Introduction to C++ Programming Language.* Preuzezo 04.08.2023 s https://www.geeksforgeeks.org/introduction-to-c-programming-language/

David Watson [slika] (2020). *ESP32 Pinout, Datasheet, Features & Applications*. Preuzeto 05.08.2023. s https://www.theengineeringprojects.com/2020/12/esp32-pinout-datasheet-features-applications.html

Electroduino [slika] [blog] (2020). *LDR sensor module | How LDR Sensor Works*. Preuzeto 07.08.2023. s https://www.electroduino.com/ldr-sensor-module-how-ldr-sensor-works/

Handsome Technology [slika] [user guide] (n.d.) *2 Channel 5V Optical Isolated Relay Module.* Preuzeto 09.08.2023. s http://www.handsontec.com/dataspecs/2Ch-relay.pdf

LarryD [slika] [forum post] (2022). *Review of an ESP32 project schematic*. Preuzeto 05.08.2023. s https://forum.arduino.cc/t/review-of-an-esp32-project-schematic/984061

Martin Heller (2022). *Visual Studio vs. Visual Studio Code: How to choose*. Preuzeto 03.08.2023. s https://www.infoworld.com/article/3436860/visual-studio-vs-visual-studio-code-how-to-choose.html

Martin Heller (2022). *What is Visual Studio Code? Microsoft's extensible code editor*. Preuzeto 03.08.2023. s https://www.infoworld.com/article/3666488/what-is-visual-studio-code-microsofts-extensible-code-editor.html

Mouser Electronics [slika] (n.d.) *DHT11 Humidity & Temperature Sensor*. Preuzeto 07.08.2023. s https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

Python Software Foundation (n.d.) *What is Python? Executive Summary.* Preuzeto 04.08.2023. s https://www.python.org/doc/essays/blurb/

Raspberry Pi Foundation [slika] (n.d.) *Raspberry Pi hardware*. Preuzeto 09.08.2023. s https://www.raspberrypi.com/documentation/computers/raspberry-pi.html

Raspberry Pi Foundation (n.d.) *What is Raspberry Pi*. Preuzeto 07.08.2023. s https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/

6. Popis slika