Internet stvari

- Ispitni zadatak -

Izveštaj o projektu “**Pametni dom**”

**Univerzitet Singidunum**

**Tehnički fakultet, Softversko i informaciono inženjerstvo**

|  |  |
| --- | --- |
| **Profesor**:  prof. dr Marko Tanasković | **Student**:  Stefan Dimitrijević 2018200497 |
| **Asistent**:  Uroš Dragović |

10.05.2022, Beograd

[1. Uvod u problem i zahteve projekta 3](#__RefHeading___Toc300_135993676)

[2. Rešenje problema 4](#__RefHeading___Toc302_135993676)

[2.1 Opšte ideja rešavanja problema 4](#__RefHeading___Toc304_135993676)

[2.2 Implementaciono rešenje 5](#__RefHeading___Toc306_135993676)

[2.2.1 Arduino 5](#__RefHeading___Toc308_135993676)

[2.2.2 Python Web server 11](#__RefHeading___Toc310_135993676)

[3. Šematski prikaz povezivanja 17](#__RefHeading___Toc312_135993676)

[3.1 Simulator 17](#__RefHeading___Toc314_135993676)

[4. Izgled grafičkog korisničkog interfejsa 18](#__RefHeading___Toc403_1339875447)

[5. Korišćeni materijali 19](#__RefHeading___Toc316_135993676)

# **1. Uvod u problem i zahteve projekta**

U sledećem izveštaju biće opisan problem ispitnog zadatka ‚‚**Pametnog doma**“, kao i rešenje datog problema.

Kada je reč o samim zahtevima Arduino sistema, oni su sledeći:

1. Pametni dom treba da omogući kontrolu svetla(u projektu je svetlo unutar doma realizovano pomoću jednog ***LED***-a) putem fizičkog prekidača(tzv. ‚‚*slider*“), kao i digitalnim putem – poslaće se digitalni signal sa Arduino mikrokontrolera koji će pomerati kotvu na releju. Takođe, ova dva načina kontrola svetla treba da su nezavisna jedno od drugog; ako se svetlo aktivira relejem, prekidačem će se ugasiti i obrnuto – ako se svetlo aktivira prekidačem, treba da smo u mogućnosti da bi trebalo promenom stanja releja da isključimo svetlo.
2. Osim svetla, vrši se i *PWM* kontrola sistema za ventilaciju – specificira se procentualna vrednost brzine kojom se ventilator okreće tako što se ili vrše očitavanja sa potenciometra, namenjenog za kontrolu ventilacije, ili se Arduino mikrokontroleru prosledi odgovarajuća poruka koja vrši ovu kontrolu.
3. Sistem takođe sadrži mehanizam za otvaranje vrata, koristeći servo motor; ukoliko se detektuje objekat na daljini manjoj od 5 cm, otvaraju se vrata, u suprotnom se vrata zatvaraju. Za detekciju udaljenosti objekata koristi se ultrasonični senzor.
4. Sistem takođe treba nekom Web serveru da šalje izveštaje sa senzora osvetljenja i temperature svakog minuta, kao i da šalje ukupan broj otvaranja vrata i aktivacije releja.

Za realizaciju projekta korišćen je ***Arduino*** mikrokontroler, senzori za merenje temperature, osvetljenja, udaljenosti, kao i brojne aktuatorske komponente: motori, releji… Povezivanje komponenti izvršeno je pomoću *breadboard*-a i ‚‚*džamper*“ kablova. Za razvoj i upload koda na *EEPROM* mikrokontrolera korišćen je *Arduino IDE*.

Što se tiče aplikaciono-interfejsnog dela, zahtevi su sledeći:

1. Aplikacija treba da komunicira sa ***ThingSpeak*** servisom koji će se koristiti za skladištenje merenja, kao i čitanje istih radi pisanja izveštaja na specificiranu eMail adresu;
2. Treba napraviti Web server, kao i odgovarajuće interfejse koje korisnik može koristiti, za udaljenu kontrolu Arduina – kao što je već rečeno, potrebno je kontrolisati svetlo(relej), brzinu ventilatora, kao i otvaranje/zatvaranje vrata;
3. Veb server treba da prikaže merenja sa senzora koje mu Arduino prosleđuje;
4. Jednom dnevno je potrebno poslati izveštaj na mail adresu; ovaj izveštaj treba da sadrži prosečnu vrednost temperature i osvetljenosti, kao i ukupan broj otvaranja vrata i promena stanja releja

Za realizaciju aplikacionog dela projekta korišćen je *Python* programski jezik, koristeći *Flask* kao *framework* za ‚‚*backend*“, pored toga, korišćeni su i pomoćni paketi za rad, koje je potrebno preuzeti pomoću *Python* menadžera paketa(***pip***):

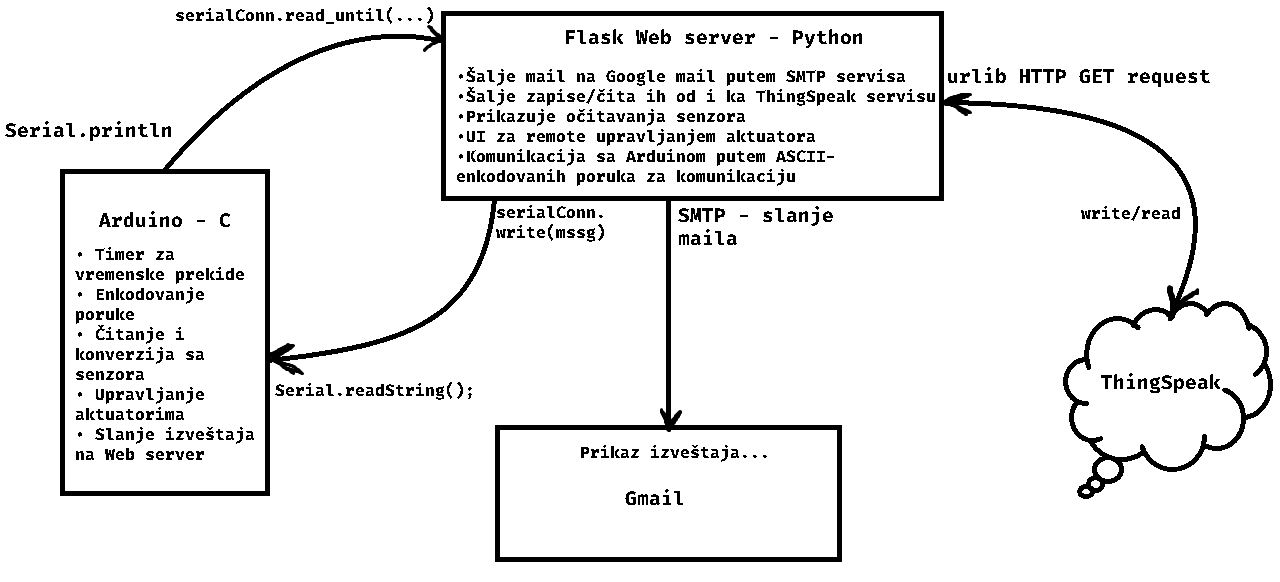
1. ***pyserial*** – koji koristimo za serijsku komunikaciju sa našim mikrokontrolerom;
2. ***requests*** – koristimo da ThingSpeak-u šaljemo ***HTTP*** zahteve za čitanjem/pisanjem;

# **2. Rešenje problema**

U sledećem delu opisaćemo detaljno kako se rešava opisani problem

## **2.1 Opšte ideja rešavanja problema**

Opšta šema načina komunikacije data je na sledećoj slici:

Slika 1. Šema komunikacije komponenti

Senzori i aktuatori su povezani sa Arduino mikrokontrolerom, a Web server je, kao *Python* skripta, pokrenut na portu 5000; omogućava korisniku određeni grafički korisnički interfejs putem kojim može da deluje na određene aktuatore povezane na Arduino, server po automatizmu u pozadini takođe prima povremene jednominutne izveštaje sa Arduina i šalje ih na ThingSpeak, a takođe i jednom dnevno šalje izveštaj na mail, čitajući podatke skladištene na ThingSpeak cloudu.

Arduino, putem vremenskih prekida, periodično(svakih 60 sekundi) šalje Web serveru izveštaje o temperaturi, osvetljenju, broju otvaranja vrata i releja, naravno, ove vrednosti mora da konvertuje u odgovarajuci oblik buduci da ima i analognih i digitalnih senzora… Na osnovu vrednosti senzora, šalje upravljačke signale ka aktuatorima, ili, ukoliko je išta poslato, prima podatke od Web servera putem serijske veze i onda upravlja aktuatorima…

Web server je zadužen da osluškuje za bilo kakve poruke koje mu mikrokontroler prosleđuje, ažurira vrednosti senzora i prikazuje ih korisniku, kao i da korisniku da mogućnost da putem grafičkog korisničkog interfejsa upravlja svetlom, vratima i brojem obrtaja ventilatora. Takođe, u pozadinskim nitima, Web server šalje periodična očitavanja sa senzora, a na svaka 24 h na mail šalje izveštaj koji sadrži agregirane podatke.

## **2.2 Implementaciono rešenje**

Kod mikrokontrolera je pisan u posebnoj i prilagođenoj verziji jezika C za Arduino. Korišćene su i neke dostupne biblioteke koje olakšavaju implementaciju mnogih funkcionalnosti.

Kod Web servera je pisan u Python programskom jeziku. Korišćen je *Flask* *framework*, *ThingSpeak* API za komunikaciju sa tom platformom, Google mejl (*Gmail*) za slanje i prijem izveštaja. Za asinhrono slanje podataka korišćen je *fetch API* unutar *JavaScripta*.

Za razvoj korišćeno je Arduino integrisano razvojno okruženje koje isporučuje editor teksta, debager, kompajler koda i programator za upload koda na EEPROM mikrokontrolera.

### **2.2.1 Arduino**

Na početku se uključuju biblioteke i definišu globalne konstante i promenljive

Slika 2. Definicija konstanti I globalnih promenljivih

Pre svega, naredbama pretprocesora, uključuju se direktive neophodne za rad – ***TimerOne*** za vremenski prekid i ???…

Zatim se, koristeći opet naredbe pretprocesora, definišu konstante koje mapiraju neka čitljiva imena na analogne i digitalne portove Arduina.

Zatim, sledećim redosledom, definišemo promenljive koje koristimo tokom rada:

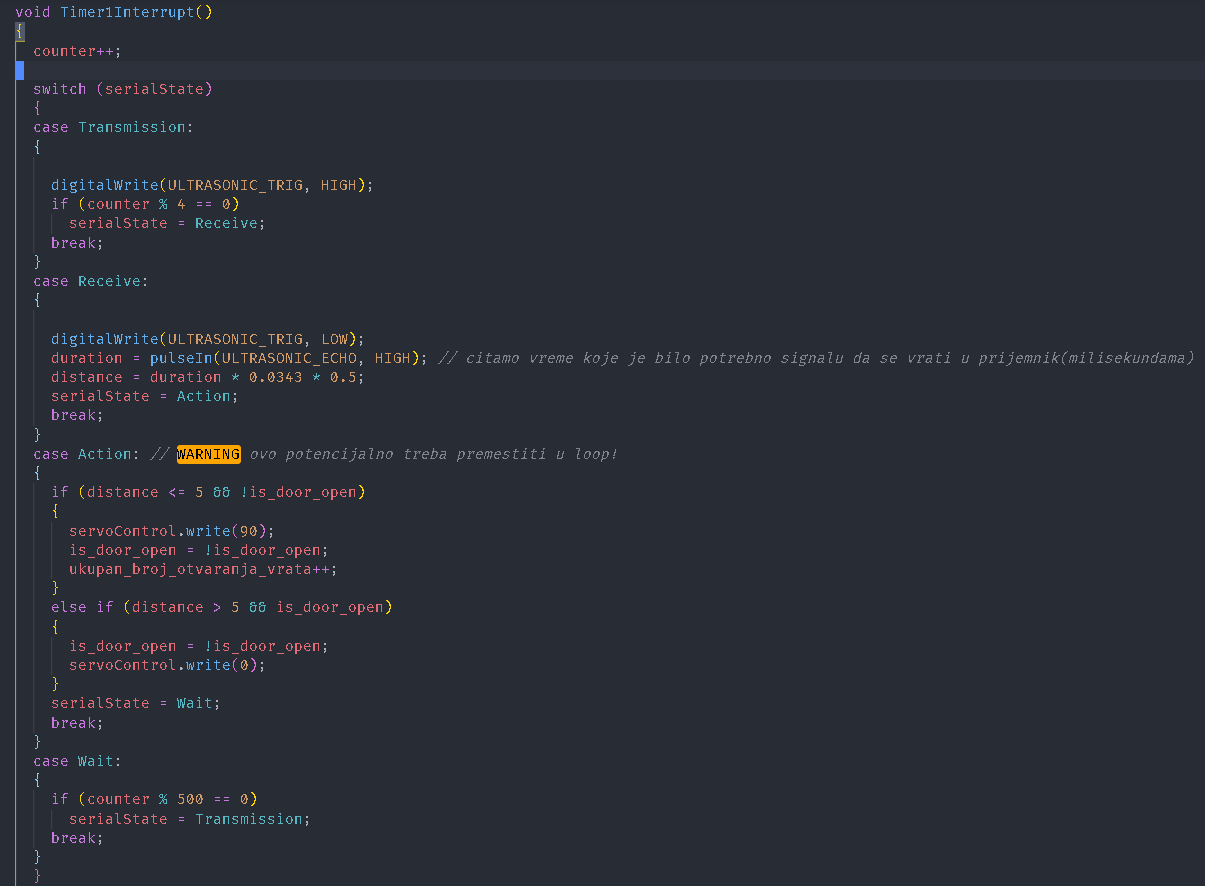
1. ***RELAY\_STATE –*** inicijalizujemo stanje releja i ovu promenljivu koristimo da radimo *toggle* stanja releja(otvoreno-zatvoreno);
2. ***vrednost\_potenciometra –*** nova vrednost koju učitavamo sa analognog ulaza gde je potenciometar povezan;
3. ***stara\_vrednost\_potenciometra*** – stara učitana vrednost potenc. Koristimo je da bi izbegli stalni upis PWM signala nad elektromotorom, što poboljšava performanse;
4. ***otpornost\_otpornika*** – otpornost otpornika kojeg koristimo kao razdelnik napona, u kombinaciji sa fotootpornikom, da bismo očitali napon i konvertovali ga u količinu svetlosti
5. ***TIMER\_DURATION –*** specificira vreme(u mikrosekundama) na koliko će se okidati vremenski prekid;
6. ***counter*** – broji koliko puta se okinuo vremenski prekid, ovo koristimo da bismo znali koliko je vremena tačno prošlo;

Zatim definišemo *enumeration* tip podatka koji ima četiri vrednosti, svaka od ovih vrednosti oslikava stanje našeg serijskog ulaza/izlaza, a koje ćemo koristiti pri merenju.

1. **TRANSMISSION** – stanje u kojem se sa transmitera šalje signal koji se odbija od udaljene površine i pada na prijemnik u nekom trenutku;
2. **RECEIVE** – stanje u kojem se čita vrednost na prijemniku kako bi se odredilo vreme koje je prošlo od trenutka kada je poslat signal sa transmitera;
3. **ACTION** – stanje u kojem se po potrebi zadaju komande aktuatorima;
4. **WAIT** – stanje koje predstavlja malo odlaganje ponavljanja, odnosno ponovnog prolaska kroz funkciju mašine stanja.

Nakon toga deklarišemo sledeće promenljive:

* + 1. ***duration*** – vreme koje je bilo potrebno signalu da se vrati u prijemnik;
    2. ***distance*** – udaljenost objekta od ultrasoničnog senzora;
    3. ***ukupan\_broj\_promena\_stanja\_releja*** – koliko puta je relej promenio svoje stanje;
    4. ***ukupan\_broj\_otvaranja\_vrata*** – koliko puta su vrata bila otvorena
    5. is\_door\_open – da li su vrata trenutno otvorena(inicijalno ne), koristimo iz istog razloga kao i ***star\_vrednost\_potenciometra***.
    6. ***servoContro***l– objekat iz biblioteke ***Servo.h*** kojeg je neophodno inicijalizovati kao globalni objekat pre korišćenja;
    7. ***serialState*** – inicijalizujemo naš *enumeration* tip

Slika 3. Vremenski prekid deo 1

Na sledećoj slici prikazan je prvi deo vremenskog prekida – tj. ***Timer1Interrupt*** funkcije.

Svaki put kada se okine funkcija(svakih 0.002 s) brojač se inkrementira za jedan. Pošto je u ***setup*** funkciji, koja će biti objašnjena kasnije, naš *enumeration* je u njoj inicijalno setovan na ***Transmission*** stanje.

Ova stanja zapravo koristimo da izvršimo periodična merenja sa ultrasoničnog senzora, i ukoliko je objekat na manje od 5 cm, otvaraju se vrata, u suprotnom su zatvorena.

Senzor funkcioniše na sledeći način: uzima se najveca moguca udaljenost koju senzor moze da izmeri, zatim se na osnovu toga izracuna vreme potrebno da signal udari u objekat.

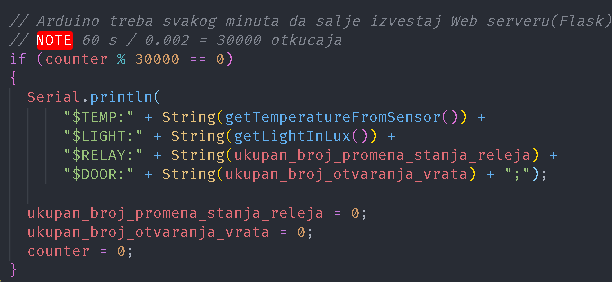
Ako je max. udaljenost 3 m(kao kod naseg uredjaja), a brzina zvuka u vazduhu 343onda je t = = 3/343 ~ 0.008746 s = 8746 µs…

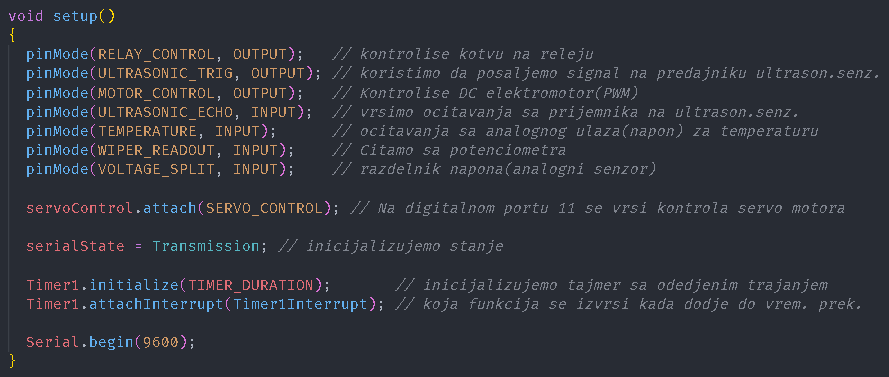
Znaci potrebno je da 8746 mikrosekundi drzimo signal na *HIGH*, a nakon toga da ga ugasimo i da na prijemnoj strani izvrsimo citanje *HIGH* signala, tj. koliko on traje(u mikrosekundama).

Buduci da smo stavili da se tajmer okida na 2000 mikrosekundi, da bismo dosli do  
priblizne vrednosti od 8000 mikrosekundi vremenski prekid treba da se okine 4 puta jer 4\*2000=8000 mikrosekundi, nakon toga treba da setujemo signal na *LOW*.

U stanju ***Transmission*** mi držimo signal na *HIGH* ~8000 mikrosekundi, nakon čega prelazimo u stanje ***Receive***, gde signal setuje na *LOW* I osluškujemo, koristeći ***pulseIn***, koliko dugo traje signal *HIGH* na prijemniku. Nakon što dobijemo ***duration*** jednostavnom primenom formule izračunamo distancu objekta, podeljenu sa dva, jer nama treba pređeni put u jednom smeru(a ne i u povratku).

U stanju ***Action*** samo detektujemo da li je objekat dovoljno blizu da se vrata otvore, nakon čega se prelazi u stanje ***Wait*** koje čeka jednu sekundu, nakon čega opet se vraćamo u stanje ***Transmission***.

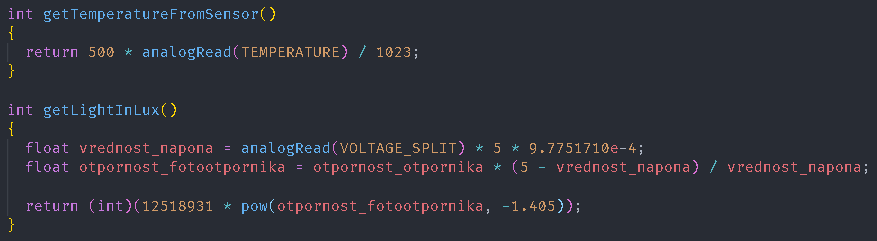
Slika 4. Vremenski prekid deo 2

U ovom delu vremenskog prekida se šalje konkretni izveštaj Flask serveru putem serijske komunikacije, kao što je opisano u komentaru, svaki put kada ***counter*** otkuca 30000 otkucaja, to znači da je prošlo 60 sekundi, odnosno 1 minut, nakon čega se podaci pakuju u određen format poruke, a vrednosti se reinicijalizuju na početne i ceo ciklus se ponavlja.

Slika 5. Setup funkcija

Setup funkcija služi za inicijalna podešavanja, kao što su definisanje modova pinova, dodeljivanje početnih vrednosti nekih promenljivih ili postavljanje početnih vrednosti nekih pinova (aktuatora). Ovde se takođe inicijalizuje funkcija vremenskog prekida i postavlja vreme na koje će se ona izvršavati. Takođe se definiše ***baud rate*** na kojem naša serijska komunikacija radi.

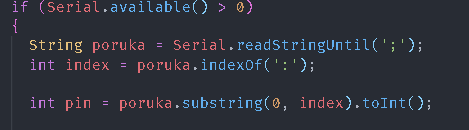
Sledeće dve funkcije se koriste za očitavanje sa senzora temperature i očitavanje osvetljenja:

Slika 6. Funkcije senzora

***getTemperatureFromSensor*** je funkcija koja očitava analogni napon sa analognog ulaza(čija je vrednost od 0-1023, gde je 0=0 V, a 1023=5 V). Određenom formulom, u zavisnosti od proizvođača senzora, dati napon konvertujemo u temperaturu. Budući da smo koristili ***LM35*** senzor formula je: .

Kao senzor osvetljenja koristi se fotootpornik čija otpornost direktno zavisi od svetlosti koja pada na njega.

Poslednja i najbitnija funkcija Arduina je ***loop***, koja se poziva u beskonačnost i izvršava od početka do kraja dokle god Arduino ima napajanje(i kod u svom *EEPROM*).

******

Slika 7. Ulaz u loop

Prvo ispitujemo da li postoje bajtovi dostupni u prihvatnom *shift* registru, na ulazu u Arduino, ukoliko postoje, čitamo ceo string dok se ne dođe do znaka „;“, jer smo tako enkodovali poruke – da se sve završavaju ovim simbolom.

Pojedinačni parametri unutar poruke razdvojeni su simbolom „:“, gde se sa leve strane nalazi ***pin*** na koji se šalje/sa kojeg se čita, dok se sa desne strane nalazi konkretna vrednost(koja je opcionalana, u slučaju da ne očekujemo bilo kakav prijem – što je slučaj kod *LED*).

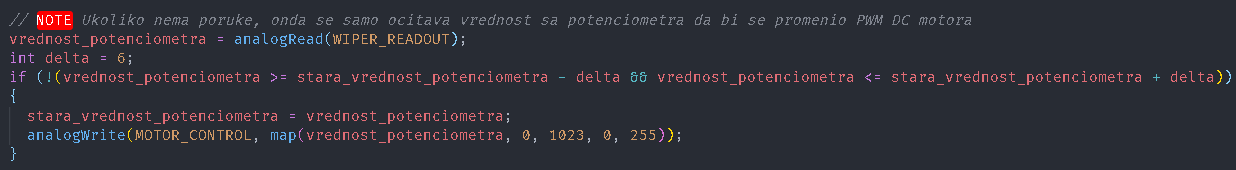
Slika 8. switch-case koji određuje na koji uređaj se šta odnosi

Ukoliko se poruka odnosi na (de)aktivaciju LED koristi se format “4:;”, ovo ce samo uraditi *toggle* diode – tacnije (de)aktivirace se relej koji je kontrolise.

Ukoliko se serijska poruka odnosi na kontrolu motora, koristi se pin 5 I poruka formata “5:0-255;”, gde 0-255 oznacava PWM vrednost koju saljemo na motor, tj. Kojom brzinom ce se vrteti, gde je 0 = 0% obrtaja, a 255 = 100% obrtaja.

Ako zelimo da vršimo remote kontrolu servo motora, na frontu imamo mogućnost da otvorimo ili zatvorimo vrata, ukoliko otvaramo vrata poruka se enkodira na sledeći način – „11:O;”, dok ako zatvaramo koristi se “11:C;”

Funkcija koja sa analognog temperaturnog senzora čita vrednosti i pretvara ih u temperaturu izraženu u stepenima Celzijusa.

U slučaju da ne postoji ništa na serijskom ulazu, izvršiće se manuelna kontrola elektromotora putem potenciometra

Slika 9. Očitavanje potenciometra

Pre svega očitava se analogna vrednost napona sa potenciometra, takođe čuvamo i podatke o staroj vrednosti, iz prethodno pomenutih razloga o performansama, takođe, budući da je u realnosti potenciometar nesavršena komponenta i da konstantna očitavanja osciluju, nisu konstantna(za konstantni položaj klizača), potrebno je definisati neku ***deltu*** koja će odbacivati promene u okviru neke sredine, u suprotnom bi se motor stalno menjao, iako ne diramo potenciometar, a *remote* upravljanje ne bi bilo moguće.

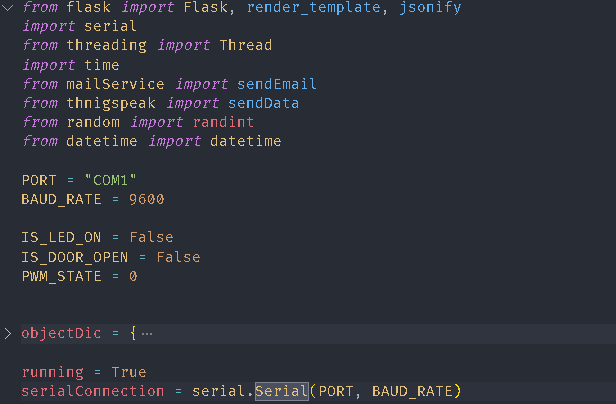
Ukoliko novoočitana vrednost nije u okrugu delte, ulazi se i menja stara vrednost(jer u sledećoj iteraciji opet može doći do oscilovanja), i nakon toga se vrši mapiranje učitane vrednosti na izlaz).

### **2.2.2 Python Web server**

**app.py**

Kod samog veb servera korišćen je *Flask* *framework* za backend, implementaciju funkcionalnosti slanja mejla putem SMTP-a i komunikacije sa *ThingSpeak* platformom putem slanja HTTP zahteva ***urlib*** bibliotekom.

Sadržaj app.py fajla iz kojeg se pokreće veb server na početku uključuje neophodne module za rad sa samim Flask okvirom, biblioteke za serijsku komunikaciju(koju je prethodno potrebno instalirati), kreiranje programskih niti(*thread*), vreme, kao i dodatni paket u koji su prethodno izdvojene skripte za komunikaciju sa *ThingSpeak* platformom i slanje mejla.

Slika 10. Inicijalizacija serverskih parametara

U početku importuju se moduli potrebni za rad, među kojima je i sam *Flask* *framework* , kao i funkcije za renderovanje *view*-a na frontu, funkcija za vraćanje podataka u ***JSON*** formatu na front.

Importuje se modul za serijsku komunikaciju, uvozi se biblioteka za rad sa nitima – ***Thread***, uvoze se biblioteke za rad sa vremenom i vremenskim žigovima(***time*** i ***datetime***), kao i funkcionalnosti dva dodatna fajla koje smo definisali – ***thnigspeak*** fajl koji sadrži ***sendData*** funkciju koja šalje podatke na ***ThingSpeak***, kao i ***mailService*** fajl koji sadrži funkciju za slanje mail-a.

Nakon toga, inicijalizuje se *PORT* na kojem je Arduino priključen sa računarom, kao i *BAUD\_RATE* na kojem radi naša serijska komunikacija(u Arduinu podešena na 9600, ovde moramo istu da ostavimo). Ova dva parametra koristimo da inicijalizujemo serijsku vezu, čiju referencu čuvamo u ***serialConnection*** promenljivoj.

Preostale promenljive se koriste pri modifikaciji vrednosti na frontu:

1. ***IS\_LED\_ON*** – da li je podrazumevano LED uključen;
2. ***IS\_DOOR\_OPEN*** – da li su vrata podrazumevano otvorena;
3. ***PWM\_STATE*** – koji je inicijalni PWM signal koji šaljemo motoru;
4. ***objectDic*** – Python dictionary tip podatka koji sadrži naše podatke koje, u okviru *JSON* formata, šaljemo na frontend asinhrono

Sledeći deo koda bavi se podešavanjem serijske komunikacije

Slika 11. Podešavanja serijske komunikacije

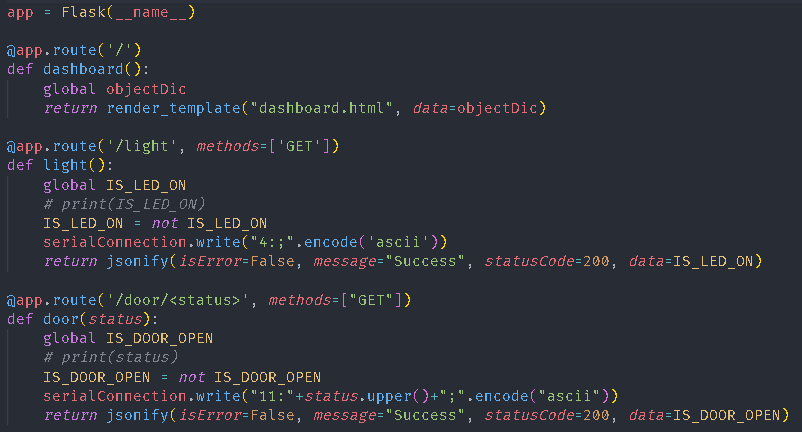
Pošto se serijska komunikacija može odvijati u bilo kom trenutku, a da ne bi blokirali glavnu nit servera, potrebno je da se ova komunikacija odvija u jednoj niti koja se izvršava beskonačno – ***threadReceiver***. Funkcija koja oslikava našu nit jeste ***receive*** i prima jedan argument – referencu ka serijskoj komunikaciji koju smo prethodno opisali.

Ukoliko ima bajtova pristiglih na prijemnoj strani( ***in\_waiting > 0*** ), potrebno je izvršiti procesuiranje poruke – prvo je čitamo dok ne stignemo do znaka „;“, a prethodno smo pomenuli da to označava delimiter naših poruka, zatim je potrebno učitani niz bajtova dekodovati kao ***ASCII*** simbole, jer Arduino ovo ne može da radi, već šalje sirov niz bajtova koji sami po sebi, bez enkodovanja, ništa ne predstavljaju. Kada poruku dekodujemo kao ***ASCII*** simbole, znamo da se radi o *string* tipu podataka u *Python*-u.

Nakon toga, ova poruka se deserijalizuje unutar funkcije ***processMessage*** koja sada izvršava proces sličan Arduinu, samo u suprotnom smeru – potrebno je ekstrahovati vrednosti koje su pojedinačni senzori poslali, kada izvršimo ekstrahovanje, ažuriramo naš objekat koji ove vrednosti čuva – ***objectDic***, a zatim, koristeći funkciju ***sendData***, koju smo definisali u posebnom fajlu, šaljemo podatke ka *ThingSpeak* servisu.

Takođe, funkcionalnost slanja mejlova, koja je periodična(svakih 24 h), se izvršava u okviru niti.

Nakon svega ovoga, inicijalizujemo ***Flask*** aplikaciju i definišemo ***endpointe*** naše aplikacije.

Slika 12. Inicijalizacija Flask aplikacije deo 1

Pre svega inicijalizujemo *Flask* aplikaciju sa određenim imenom i čuvamo referencu u određenoj promenljivoj – ***app***.

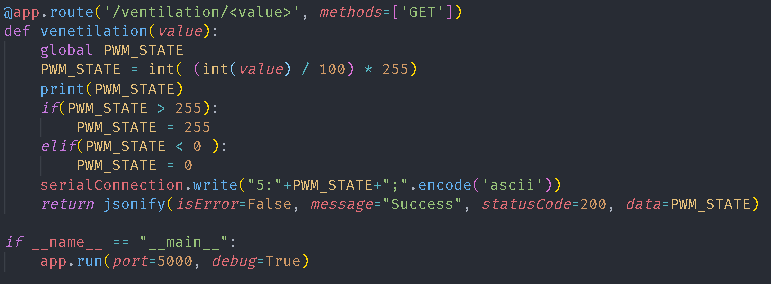
Nakon toga definišemo pojedinačne endpointe i funkcije koje se izvršavaju kada pristignu zahtevi na određene rute.

Za ovo se koristi ***@app.route(‘ruta’, methods=[niz HTTP metoda])*** kojom se specificira koja putanja će se koristiti ka resursu, kao i koje HTTP metode(za naše potrebe je ***GET*** metoda dovoljna).

Nakon toga slede pojedinačne funkcije za date endpointe, gde svaka funkcija ili vraća ***render\_template***, koja kao argumente uzima naziv HTML template-a koji će se renderovati, kao i podatke koje šaljemo tom templejtu. Alternativno možemo slati samo *JSON* format podataka, što zahteva dodatan hendling na samom frontu.

Putanja ***light*** koristi se da uradi *toggle* *LED*-a, i na serijskom izlazu upisuje odgovarajuć format poruke.

Putanja ***door*** koristi, u zavisnosti od pretisnutog dugmeta, se da otvori, odnosno zatvori vrata koristeći poruke „11:O;” i „11:C;” respektivno.

Slika 13. Inicijalizacija Flask aplikacije deo 2

Preostala putanja nam se koristi za PWM kontrolu DC motora, prvo što radimo jeste da vršimo konverziju procenata (od 0-100%) u PWM vrednost koju Arduino prihvata (0-255), ukoliko korisnik unese vrednost veću ili manju od dozvoljene, setovaće se 255 ili 0 respektivno, a zatim će se odgovarajuća poruka ispisati na serijski izlaz – „5:0-255;”.

Nakon toga samo, koristeći ***run*** metodu, pokrećemo ***Flask*** aplikaciju na portu 5000.

Preostalo je samo da objasnimo fajlove koji se bave komunikacijom sa *ThingSpeak* servisom, kao I mail servis.

Komunikacija sa *ThingSpeak*-om je krajnje jednostavna, svodi se na upućivanje običnih *HTTP GET* zahteva ka određenom URL-u, uz određene propratne parametre:

Slika 14. ThingSpeak fajl

Imamo tri varijable, gde svaka specificira parametre koje ThingSpeak nudi korisniku, ***READ\_KEY*** je API ključ koji korisnik mora da pošalje u URL-u kao parametar kada vrši bilo kakva očitavanja(što se izvršava unutar ***receiveData*** funkcije), ***WRITE\_KEY*** je ključ koji korisnik mora, pored vrednosti koje se upisuju u odgovarajuća polja, da prosledi kada god vrši bilo kakva upisivanja(što se izvršava uz pomoć ***sendData*** funkcije koju smo videli u prethodnom ***app.py*** fajlu). ***CHANNEL\_ID*** je unikatna vrednost za svakog korisnika.

Slanje mejlova je takođe krajnje jednostavno, prvo koristimo funkcionalnosti prethodno opisanog fajla da sa ThingSpeaka dovučemo potrebne podatke za slanje izveštaja(***receiveData*** funkcija). Nakon toga vršimo određena izračunavanja(agregaciju) podataka da bismo dobili podatke neophodne za naš izveštaj. Nakon toga sledi unošenje pristupnih parametara(mail adresa i lozinka) korisnika koji šalje, budući da koristimo Google mail SMTP server, korisnik mora posedovati *Gmail* nalog.

Slika 15. SMTP server

Nakon što je korisnik uspešno uneo dva pristupna parametra za svoj nalog, tek onda vršimo startovanje servera(u suprotnom se može desiti da dobije *time out*), nakon startovanja servera, koristimo ***login*** funkciju da se, prethodno unetim parametrima, ulogujemo na nalog korisnika(identifikujemo kao on).

Nakon toga se samo vrši formatiranje poruke u zahtevanom formatu, format treba da je sledeći:

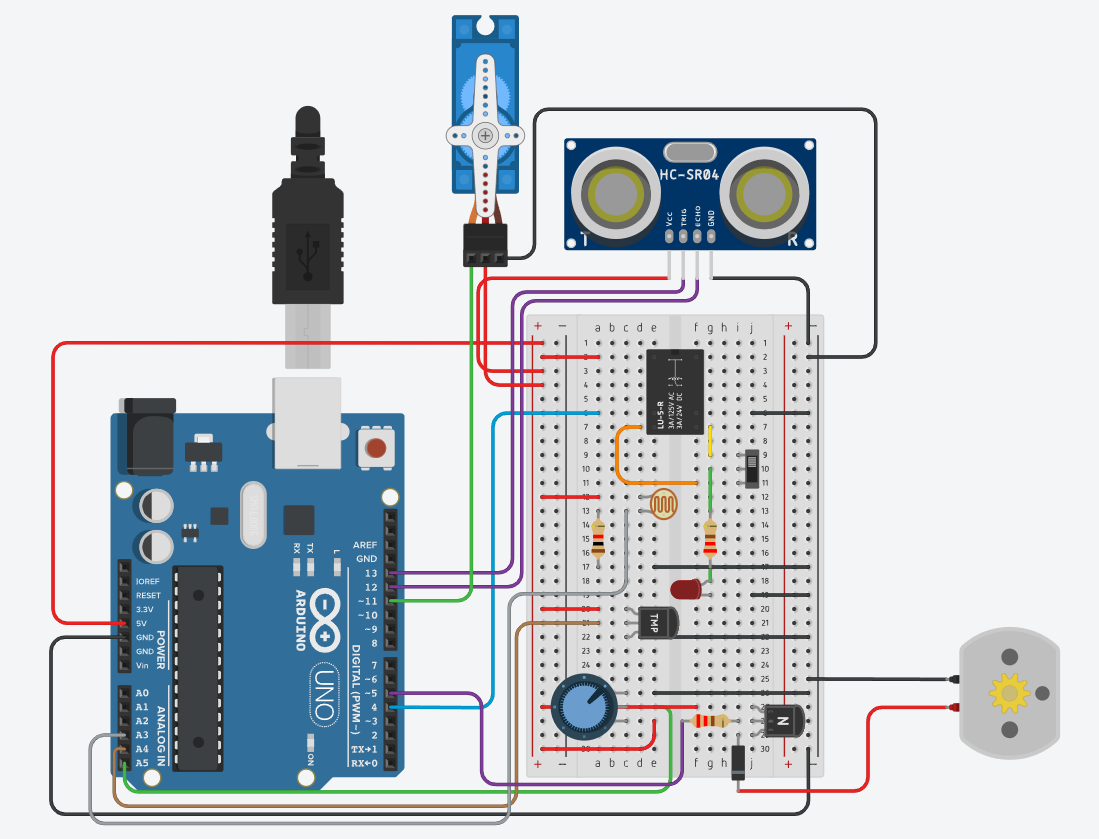
„**Subject: ovde-ide-naslov-poruke**

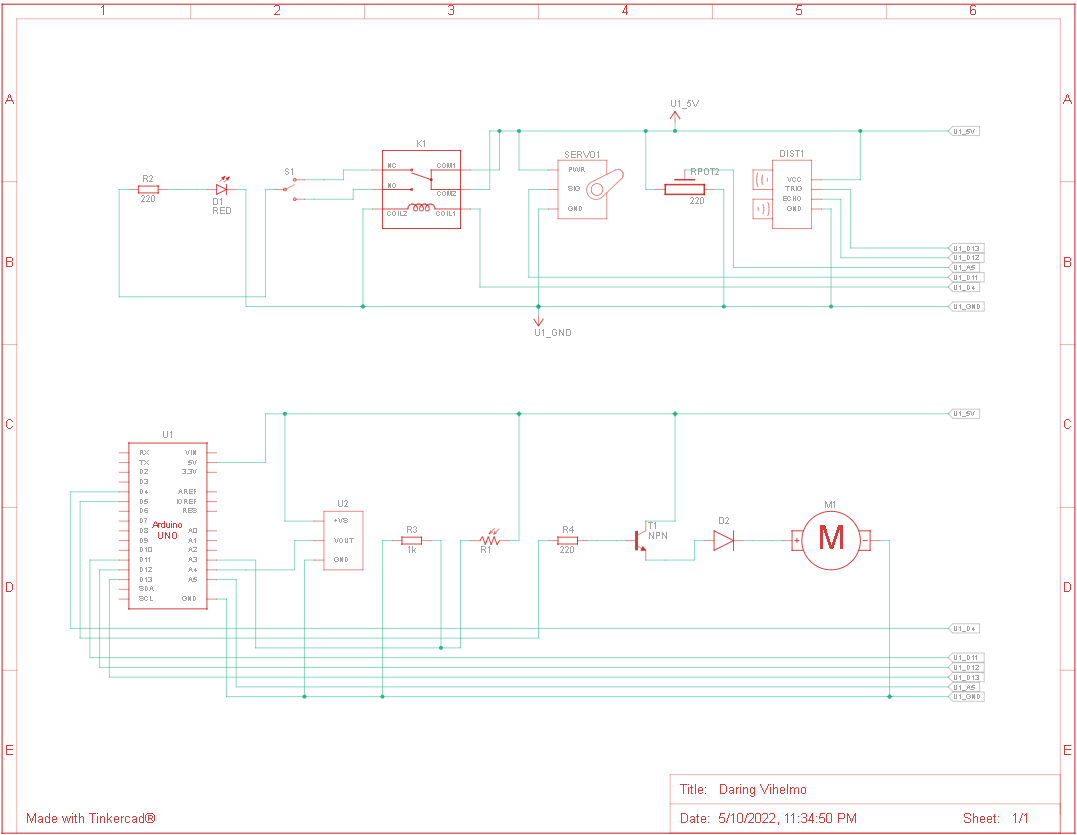
**Telo poruke ide ovde**“

Kada smo uspešno formatirali poruku, samo pomoću ***sendMail*** specificiramo sa koje adrese se šalje, na koju adresu(u ovom slučaju je adresa primaoca i pošiljaoca ista) i koja je poruka. Nakon što smo izvršili slanje zatvaramo vezu sa serverom.

# **3. Šematski prikaz povezivanja**

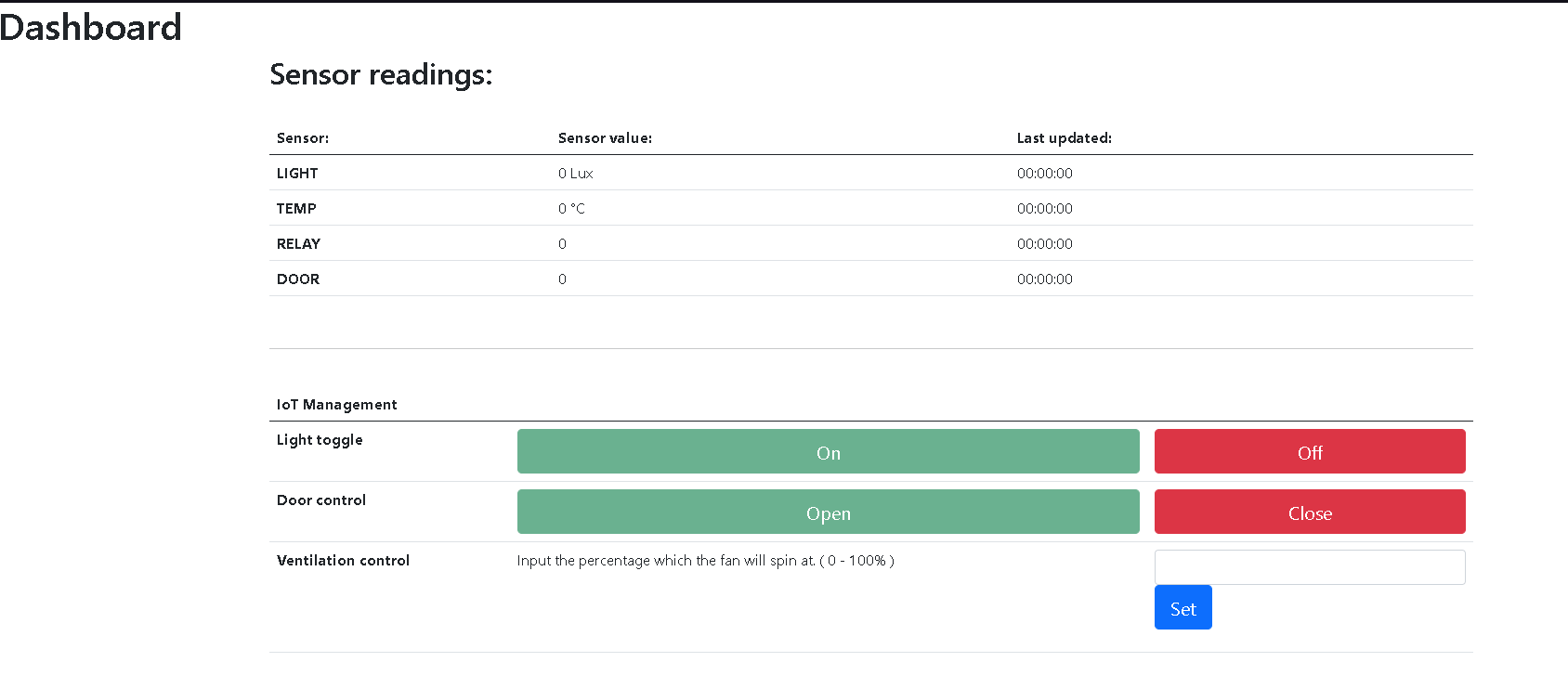
## **3.1 Simulator**

Slika 16. Grafički prikaz iz simulatora

Slika 17. Šematski prikaz iz simulatora

# **4. Izgled grafičkog korisničkog interfejsa**

Na sledećoj slici biće prikazan izgled grafičkog korisničkog interfejsa:



Slika 18. GUI

# **5****. Korišćeni materijali**

- <https://www.tinkercad.com/> (besplatni Arduino simulator)

- <https://www.arduino.cc/en/software> (besplatni, zvanični Arduino IDE)

- <https://docs.arduino.cc/> (Arduino dokumentacija)

- Marko Tanasković, 2020, *Internet stvari*, Univerzitet Singidunum, Beograd

- Materijali sa predavanja i vežbi