Sustav za automatizaciju i daljinsko upravljanje uređajima u kući temeljen na ESP32 mikrokontroleru

Šimunčić, Krunoslav

Sveučilište u Rijeci – Odjel za informatiku

Informatika - jednopredmetni

Krunoslav Šimunčić

Sustav za automatizaciju i daljinsko upravljanje uređajima u kući temeljen na ESP32 mikrokontroleru

Završni rad

Mentor: doc. Dr. sc. Miran Pobar

Rijeka, 24.9.2021.

_____

Ime i prezime studenta

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 29.3.2021.

Zadatak za završni rad

Pristupnik: Krunoslav Šimunčić

Naziv završnog rada: Sustav za automatizaciju i daljinsko upravljanje uređajima u kući temeljen na ESP32 mikrokontroleru

Naziv završnog rada na eng. jeziku: Home automation and remote control system based on the ESP32 microcontroller

Sadržaj zadatka:

Zadatak je izraditi i opisati prototip sustava za daljinsko upravljanje odabranim električnim uređajima u kući temeljen na mikrokontroleru ESP32. Opisati sklopovsku realizaciju sustava te realizaciju programske podrške.

Mentor

doc. dr. sc. Miran Pobar

Voditelj za završne radove

doc. dr. sc. Miran Pobar

Zadatak preuzet: 1.6.2021.

Šimunčić

(potpis pristupnika)

SADRŽAJ

1. SAŽETAK

U radu je opisan proces izrade sustava za daljinsko upravljanje relejima na koje su, u konkretnom slučaju, spojene LED svjetiljke u obiteljskoj kući. Sustav se sastoji od ESP32 mikrokontrolera i pripadajućih MCP23017 IO expander-a koji omogućuju kontroliranje 24 releja i 8 digitalnih 3.3V izlaza te omogućuju spajanje do 32 ulaza, odnosno tipkala iz kućne instalacije. Drugi dio sustava je web aplikacija koja omogućuje postavljanje sustava i daljinsku kontrolu. Specifičnost sustava je što omogućuje upravljanje putem Simply Dim protokola. U radu su, uz proces izrade, opisane komponente sustava, korišteni alati i programski jezici, problemi pri izradi te su navedene mogućnosti za poboljšanje sustava. Izrada sustava je bila uspješna te se sustav svakodnevno koristi.

Ključne riječi: ESP32, pametna kuća, automatizacija, daljinsko upravljanje, Python, Django, C++

2. UVOD

Posljednjih godina svjedoci smo eksponencijalnog rasta broja IoT (Internet Of Things) uređaja u svijetu. Gotovo da više ne postoji elektronički uređaj koji nema mogućnost spajanja na Internet. Ljudi nastoje što je više moguće optimizirati i ubrzati svakodnevne trivijalne radnje za što im IoT uređaji uz njihovu pripadajuću infrastrukturu daju mogućnost. No bez obzira na raširenost takvih uređaja i dalje je teško postići potpuno centralizirano upravljanje svim uređajima koje korisnik može koristiti zbog nekompatibilnosti između uređaja različitih proizvođača i sl. Ako korisnik želi putem mobilnog uređaja upaliti rasvjetu, promjeniti sobnu temperaturu, upaliti ventilaciju ili klima uređaj vjerojatno je da će za te radnje morati koristiti nekoliko aplikacija.

Cilj ovog rada je izraditi sustav pametne kuće koji će biti prilagodljiv i univerzalan. Umjesto da se uređajima u kući upravlja direktno putem proizvođačeve aplikacije sustav će upravljati nizom releja koji će kontrolirati dovod napona na uređaj. Cilj nije izraditi gotovi sustav koji može upravljati svakom vrstom uređaja već napraviti temeljni dio sustava čijom se kasnijom nadogradnjom to može postići.

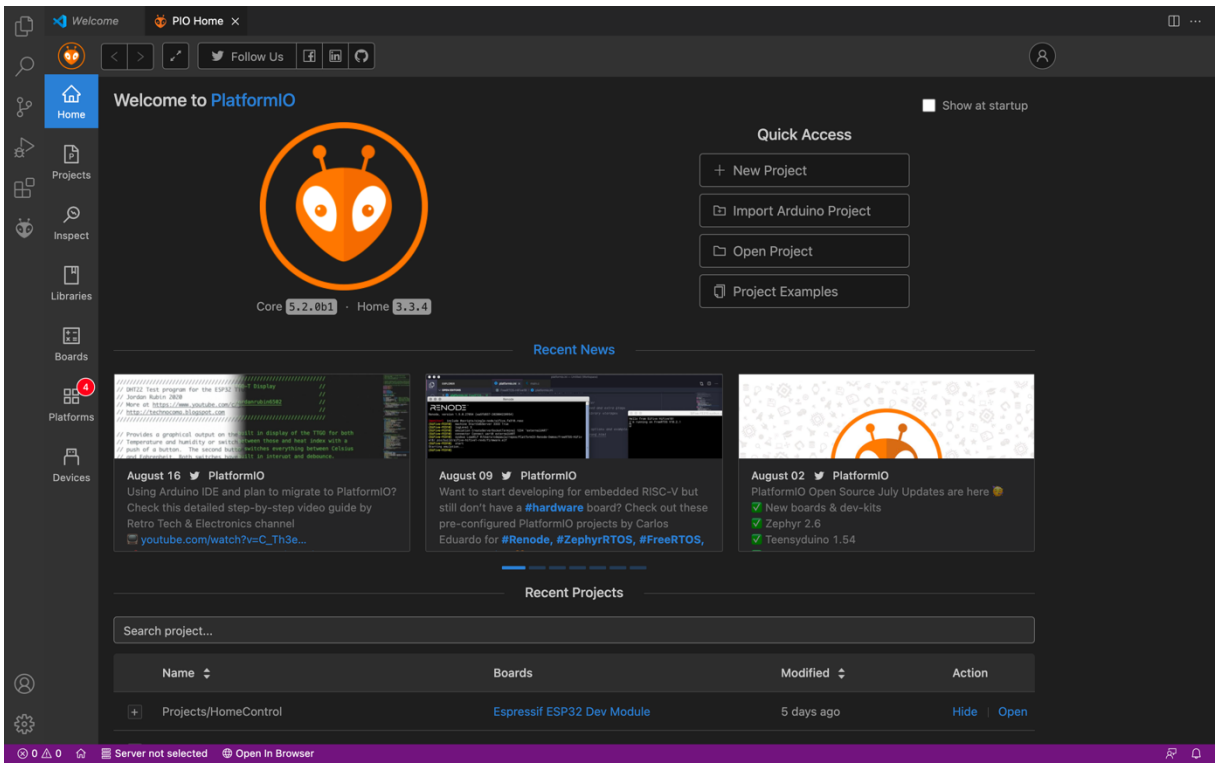
U radu se konkretno sustav implementira u obiteljsku kuću isključivo za kontrolu rasvjetom. Glavni zahtjevi sustava su da se fizičkim tipkalima upravlja grupama rasvjete koje moraju biti prilagodljive te mogućnost da se s jednog mjesta upali ili ugasi cijelokupna rasvjeta u kući no implementirane su i funkcionalnosti poput upravljanja putem interneta, upravljanja većim brojem pločica i kreiranje korisnika.

Specifičnost ovog sustava je u tome što može upravljati običnom rasvjetom ali i rasvjetom upravljanom putem Simply Dim protokola čiji se princip rada opisuje u poglavlju 4. Realizacija sustava – Hardver.

3. PRIMJENJENE PROGRAMSKE TEHNOLOGIJE I ALATI

3.1. Visual Studio Code i PlatformIO

Visual Studio Code je IDE izrađen od strane Microsofta za Windows, Linux i macOS. Nudi podršku za debugging, označavanje sintakse, inteligentno dovršavanje koda, refactoring, ugrađeni Git, itd. Korisnici mogu instalirati proširenja koja mu proširuju funkcionalnost. Jedno od tih proširenja je PlatformIO te je korišten za izradu ovog rada. PlatformIO je alat za programiranje „embedded“ sustava. Njegova decentralizirana arhitektura omogućuje programiranje gotovo svih vrsta mikrokontrolera i mikroračunala. Uz kompajliranje koda omogućuje i serijsku komunikaciju sa razvojnom pločicom te učitavanje koda u flash memoriju pločice. Nakon učitavanja koda možemo u stvarnom vremenu očitavati izlaz sa pločice putem serijske komunikacije što je posebno korisno za debugging. PlatformIO uvelike olakšava rad u odnosu na alternativna razvojna okruženja (npr. Arduino IDE) jer ima vrlo jednostavnu instalaciju vanjskih biblioteka te se zbirka raznih često korištenih biblioteka nalazi u samoj aplikaciji. Također nudi inteligentno dovršavanje koda što Arduino IDE ne nudi.



Slika 1 - Grafičko sučelje VS Code i PlatformIO

3.2. Programski jezik za ESP32 – Arduino C++

ESP32 se može programirati pomoću nekoliko programskih jezika. Neki od njih su C++, Python, Lua pa čak i JavaScript. Daleko najraširenija metoda programiranja ESP32 pločice je putem Arduino jezika. Arduino programski jezik je reducirani C++ uz neke prilagođene funkcije. U ovom radu se koristi upravo Arduino jezik iz razloga što je najraširenji te samim time ima najveću podršku u vidu vanjskih biblioteka, materijala poput gotovih primjeraka koda, riješenih problema i slično. Temelj svakog programa pisanog u tom jeziku su funkcije setup() i loop(). Sadržaj funkcije setup() se izvodi samo jednom pri pokretanju mikrokontrolera dok se nakon toga sadržaj funkcije loop() izvodi u beskonačnoj petlji. Obično se u funkciju setup() pozivaju inicijalizacijske funkcije (npr. Serial.begin() – za inicijalizaciju serijske komunikacije), pozivaju se funkcije koje je potrebno izvesti samo jednom i slično. U funkciju loop() se upisuje kod koji izvodi ono što je namjena programa. Kod treba pažljivo pisati uzimajući u obzir da se izvodi u beskonačnoj petlji.

3.3. Programski jezici za Web aplikaciju

3.3.1. HTML

HTML (HyperText Markup Language) je prezentacijski jezik za izradu web stranica. Hipertekst dokument stvara se pomoću HTML jezika. HTML jezikom oblikuje se sadržaj i stvaraju se hiperveze hipertext dokumenta (Wiki). Temeljna zadaća HTML jezika jest uputiti web preglednik kako prikazati hipertext dokument.

3.3.2. CSS

CSS (Cascading Style Sheets) je stilski jezik koji se koristi za opis prezentacije napisane pomoću HTML-a (Wiki). CSS uređuje izgled i raspored web stranice.

3.3.3. JavaScript

JavaScript je skriptni programski jezik koji se izvršava u web pregledniku na strani korisnika. JavaScript je izuzetno moćan i svestran jezik te je to razlog što ga koristi oko 97% svih web stranica na internetu. Uz pomoć AJAX metoda ili WebSocket protokola s JavaScriptom je moguće osvježiti pojedine sadržaje web stranice asinkrono bez osvježavanja cijele stranice. Za JavaScript postoje razne biblioteke (framework) kao što su Angular, React, jQuery. Također postoje biblioteke kao što su Node.js i Vue.js koje JavaScript pretvaraju u „server side“ jezik. U ovom radu se koristi u svom čistom obliku bez frameworka (tzv. Vanilla JS).

3.3.4. Python – Django framework

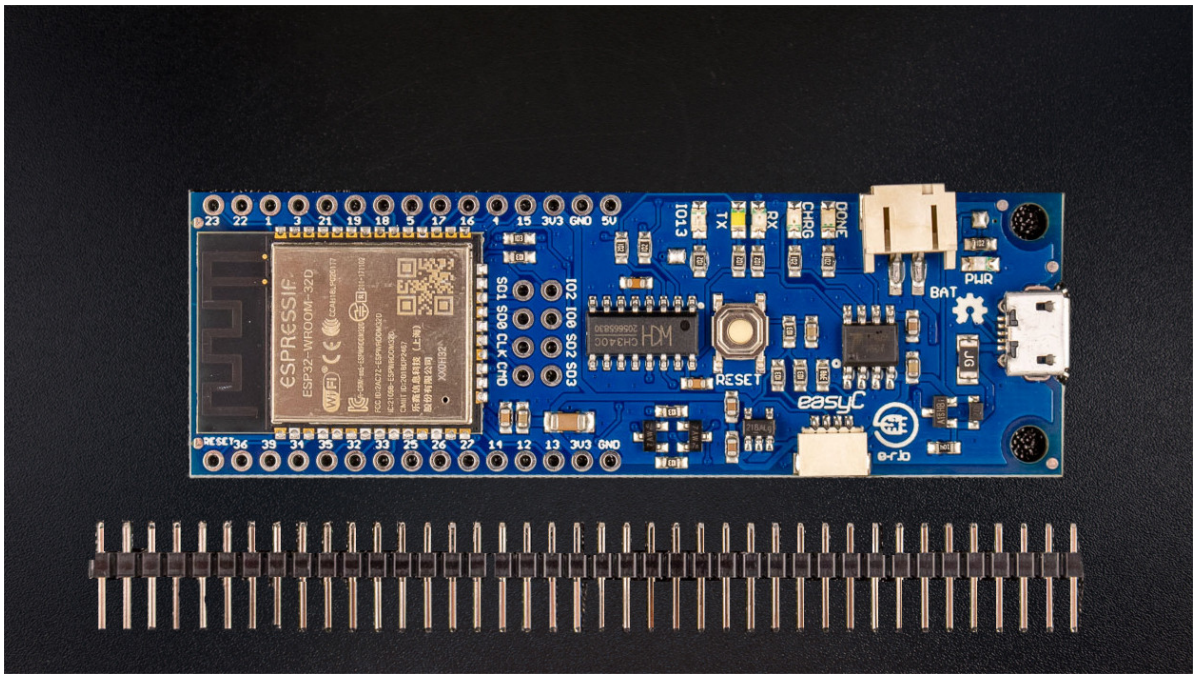
Django je besplatan i open-source Python framework. Njegov glavni cilj je stvaranje kompleksnih web aplikacija orijentiranih prema bazi podataka. Baziran je na MTV (Model Template View) arhitekturi. Django omogućuje izrazito jednostavan rad sa bazom podataka u kojem Python klasa predstavlja tablicu, a njeni atributi stupce te tablice. Sve operacije nad bazom se vrše pozivanjem na metode klasa. Django također uključuje mnogo gotovih aplikacija poput autentifikacije korisnika i administrativnog sučelja. Ima ugrađena rješenja za povećanje sigurnosti odnosno smanjenje šanse tipičnih web napada. U ovom radu Django se koristi za „back end“ dio web aplikacije.

4. HARDVERSKE KOMPONENTE SUSTAVA

4.1. Razvojna pločica Croduino NOVA32

Croduino NOVA32 je mikrokontrolerska razvojna pločica bazirana na ESP32 proizvedena od strane Hrvatske tvrtke „e-radionica“. Na pločici se konkretno nalazi Espressif ESP32-Wroom-32D modul. ESP32 nudi WiFi povezivanje, Bluetooth 4.0, 28 GPIO pinova od kojih neki podržavaju „touch“ kontrolu pomoću kapacitivnih senzora. Ima 520kB RAM memorije i 4MB flash memorije dok sam procesor ima dvije jezgre te radi na 240MHz.

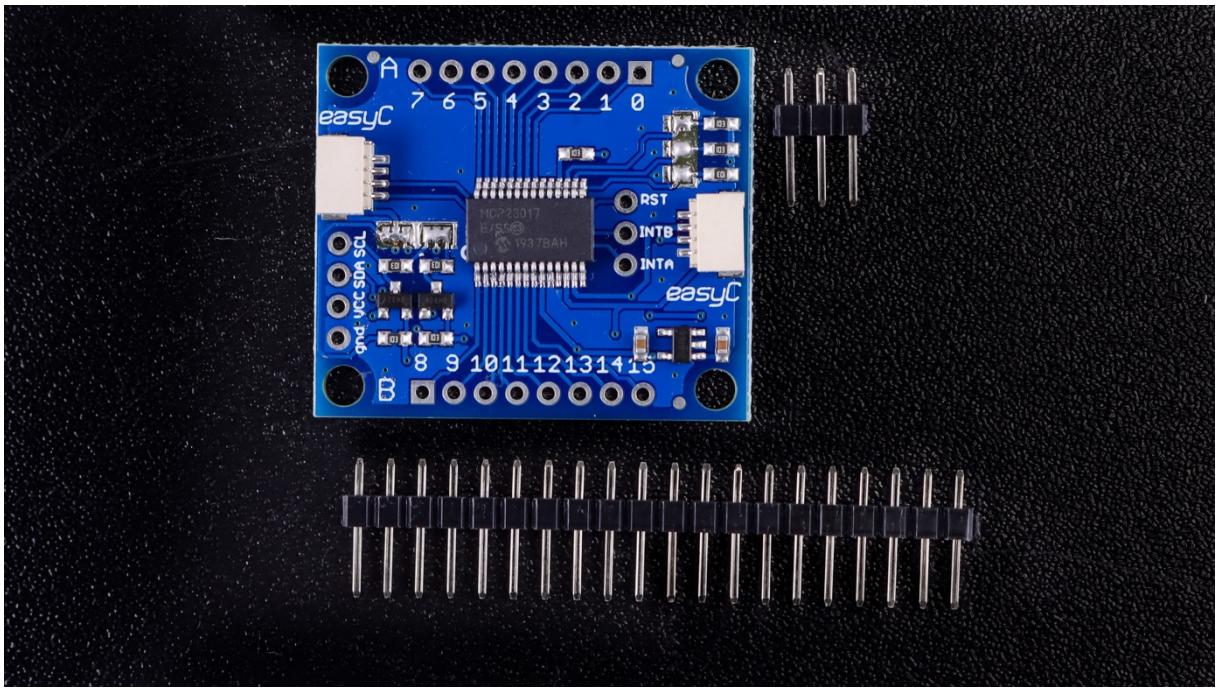
NOVA32 ima tzv. easyC konektor koji omogućuje jednostavno spajanje sa ostalim komponentama putem I^2C komunikacije. EasyC daje paralelan spoj na pinove 21(SDA) i 22(SCL), VCC(3.3V) i GND na ESP32.



Slika 2 - Razvojna pločica Croduino NOVA32

4.2. MCP23017 IO expander

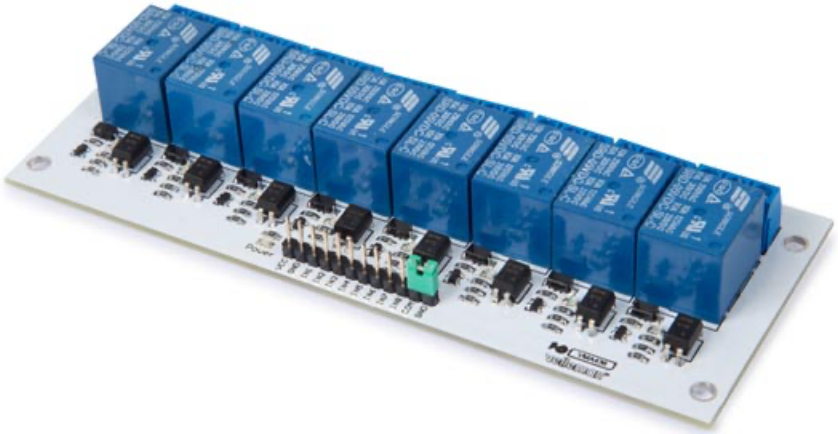
MCP23017 je uređaj koji dodaje 16 GPIO pinova mikrokontroleru putem I^2C komunikacije. Korištena pločica je također proizvedena od strane „e-radionice“. MCP23017 je povezan sa ESP32 mikrokontrolerom putem easyC kabla. U ovom radu se koristi 4 ovakve pločice te su paralelno spojene na I^2C pinove odnosno easyC konektor na ESP32.



Slika 3 - Croduino MCP23017 IO expander

4.3. Velleman 8-kanalni modul s relejima

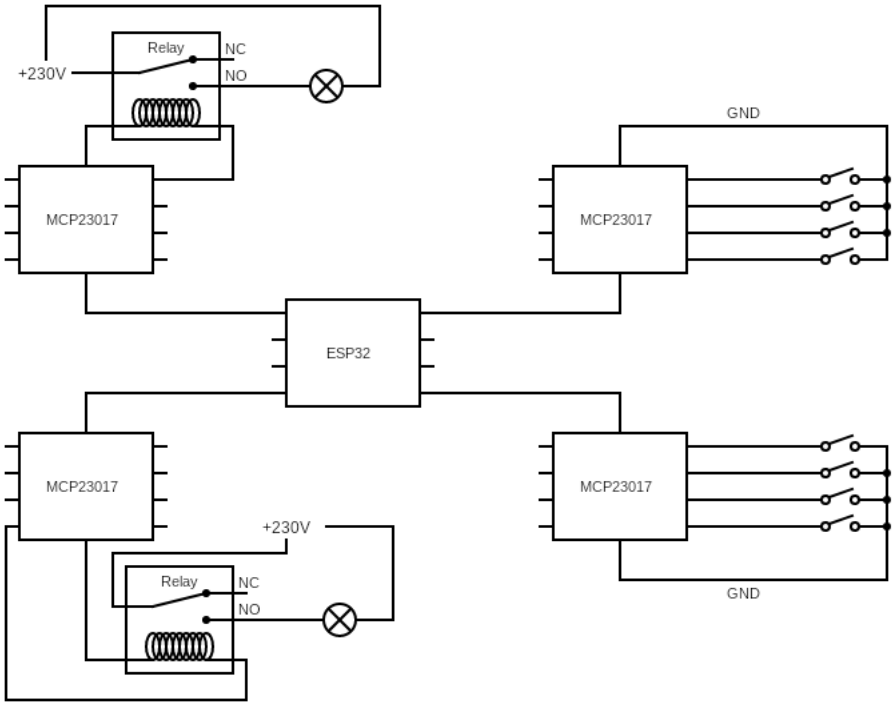
Relej je vrsta prekidača koji pomoću strujnog kruga male snage upravlja strujnim krugom veće snage. Velleman modul sa relejima je gotova pločica koja sadrži sve potrebne komponente za upravljanje relejima putem 3.3V napona. Sadrži opto-izolatore na ulazima koji galvanski odvajaju zavojnicu releja od izlaza sa mikrokontrolera. Bez toga bi induktivni napon stvoren u zavojnici releja mogao stvoriti kratak impuls u strujnom krugu te oštetiti upravljačku pločicu. Modul sa relejima iz tog razloga zahtjeva vanjsko napajanje dok se upravlja naponski preko 8 pinova koji predstavljaju releje respektivno. Upravljački strujni krug je zaseban te zahtjeva da COM pin bude spojen na GND od mikrokontrolera.



Slika 4 - Velleman 8-kanalni modul s relejima

5. REALIZACIJA SUSTAVA - HARDVER

U ovom poglavlju biti će opisana hardverska konstrukcija sustava te neki njeni specifični problemi. Slika 5 prikazuje jednostavnu shemu sustava koja predstavlja osnovni princip rada. Postoje 4 MCP23017 IO expandera od kojih su na 2 spojeni releji, a na druga 2 tipkala. Njima upravlja ESP32 mikrokontroler.



Slika 5 - Jednostavna shema sustava

5.1. Odabir načina kontrole rasvjete – DALI ili Simply Dim

U kući kojom sustav upravlja postoje 3 vrste rasvjetnih tijela: LED trake koje se upravljaju putem univerzalnog napajanja 12V i 24V, obične LED svijetiljke koje se upravljaju putem LED drivera po principu ON/OFF i LED svijetiljke varijabilne količine svijetlosti (dimmabilne) koje se upravljaju putem LED drivera koji podržava kontrolu putem „DALI“ protokola ili „Simply Dim“ načina. Rasvjetna tijela možemo generalizirati u dvije kategorije prema načinu upravljanja: obična i Simply Dim. Za kontrolu dimmabilnih svjetiljka odabran je način „Simply Dim“ no kako bi približili razloge zašto potrebno je opisati oba protokola te objasniti njihove prednosti i mane.

5.1.1. DALI protokol

Digital Addressable Lighting Interface (DALI) je otvoreni standardni protokol za upravljanje rasvjetom. Specificiran je putem nekoliko tehnoloških standarda u IEC 62386 normi. Ti standardi omogućuju da oprema od različitih proizvođača bude međusobno kompatibilna.

Ukratko, DALI protokol povezuje sva rasvjetna tijela pomoću dvije žice DA+ i DA- na sabirnicu koja se tipično nalazi na 16V. Na tu sabirnicu je povezan i tzv. DALI master koji komunicira sa rasvjetom na način da DA+ i DA- sabirnice kratko spoji te tako šalje signale prema svim ostalim uređajima u mreži (tzv. DALI slave). Svakom slave uređaju se dodijeljuje kratka adresa koja može biti od 0 do 63 (64 rasvjetna tijela je ograničenje u DALI mreži). Signali se šalju putem Manchester kodiranja. Slave uređaji mogu odgovarati te slati povratne informacije poput stanja i grešaka.

DALI komunikacija bi bila idealna za ovaj sustav jer bi mikrokontroler mogao u svakom trenutku poslati upit i znati u kojem se stanju svijetiljka nalazi i pouzdano podešavati svijetlinu.

No usprkos tome za sustav nije odabrana DALI komunikacija iz razloga što bi mikrokontroler morao slati Manchester kodirane signale što dosta povećava kompleksnost sustava, no ipak veći problem bi bio što bi uz DALI komunikaciju morali postojati releji koji bi upravljali običnom rasvjetom te struja koja teče kroz te releje stvara elektromagnetske smetnje koje su u testiranjima stvarale probleme DALI komunikaciji u vidu nekontroliranog paljenja i gašenja svijetiljki te bi se time narušila pouzdanost rada sustava.

5.1.2. Simply Dim

Simply Dim je način kontrole LED drivera putem napona električne mreže 230V. Naziv Simply Dim se nalazi u uputama LED drivera uz pripadajuću shemu spajanja no osim toga na internetu ne postoji nikakav podatak ni tehnička dokumentacija tog načina kontrole rasvjete te stoga moramo pretpostaviti da je to proizvođačev vlasnički naziv. Rasvjeta se kontrolira jednostavno na način da se na kontakte (koji su ujedno i kontakti za DALI) kod pritiska tipkala dovede impuls 230V koji pali ili gasi svijetiljku.

U slučaju da impuls potraje duže od 0.8 sekundi te da je svijetiljka ugašena u trenutku slanja impulsa, svijetiljka će se upaliti i postupno mijenjati svijetlinu dok god se na kontaktima nalazi napon ili dok svijetlina ne dođe do gornje ili donje granice.

U slučaju impulsa kraćeg od 0.8 sekundi svijetiljka će promjeniti stanje.

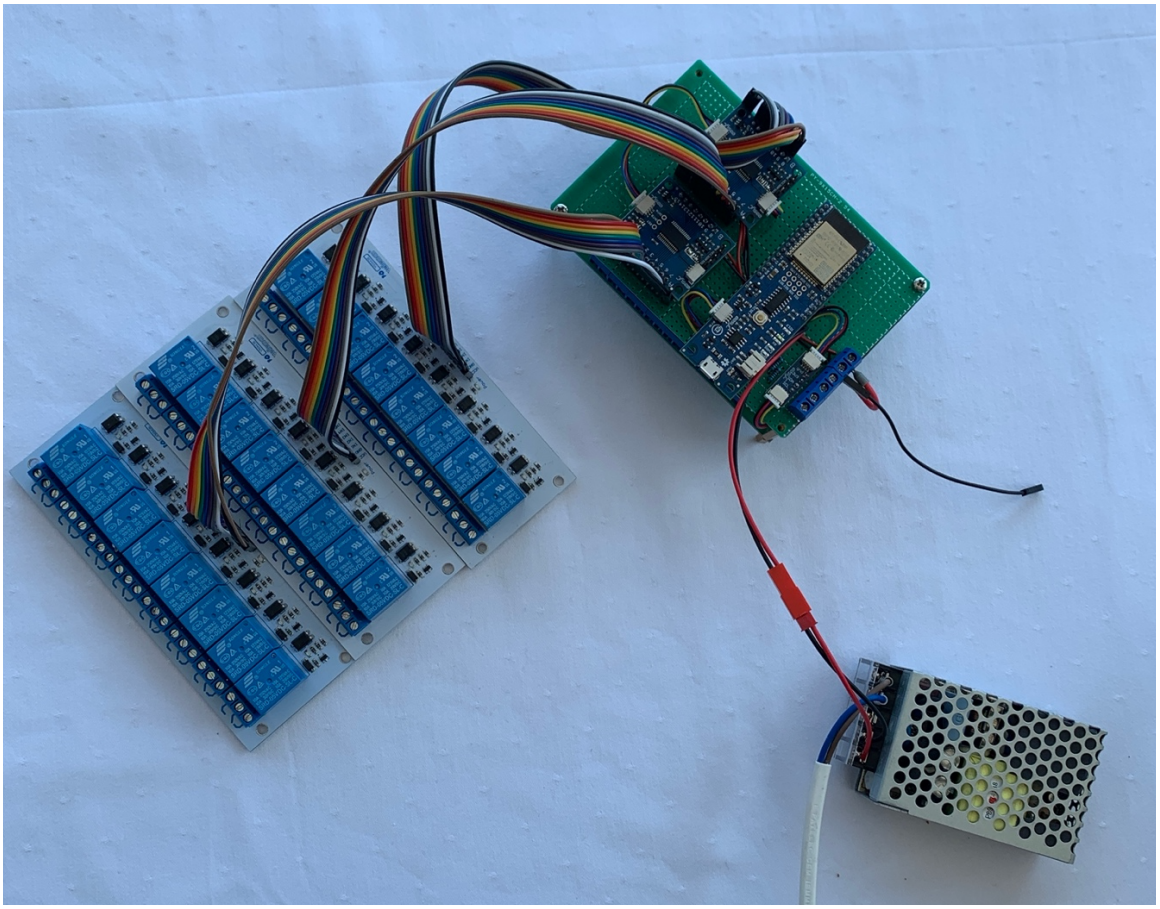
U slučaju impulsa dužeg od 0.8 sekundi dok je svjetiljka upaljena ona će mijenjati svijetlinu te će ostati upaljena.

Takvo ponašanje komplicira sustav iz razloga što mikroračunalo mora nakon svake promjene stanja provjeriti koliko je dugo tipkalo bilo pritisnuto te s obzirom na to spremiti stanje svake pojedine svijetiljke ili grupe svjetiljki. LED driveri također čak i u slučaju nestanka struje pamte zadnje stanje stoga kod ponovne uspostave napajanja mikroračunalo također mora pamtiti u kojem je stanju svaka svijetiljka.

Bez obzira na tu komplikaciju u testiranjima se Simply Dim pokazao kao pouzdaniji način upravljanja te je zato odabran. Također u rijetkom slučaju problema kao što je nejednako paljenje svijetiljki u grupi (npr. u grupi od 3 svijetiljke kod paljenja se jedna ne upali) problem se jednostavno riješi dužim držanjem tipkala koje će upaliti tu jednu svijetiljku no u mikrokontroler će ih i dalje voditi kao upaljene jer je pritisak trajao duže od 0.8 sekundi dok su bile upaljene.

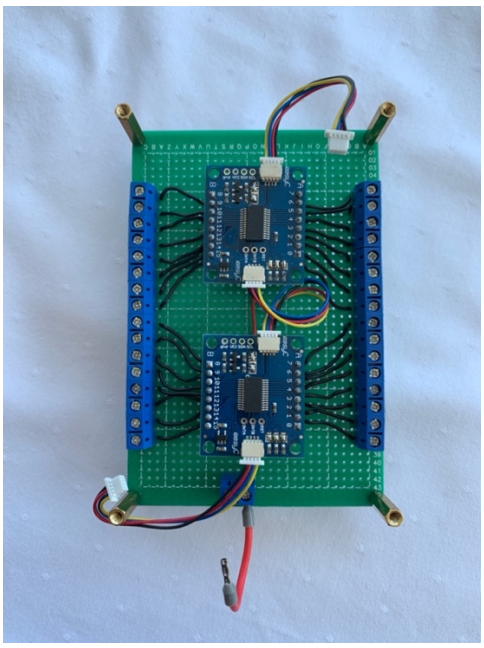
5.2. Pregled hardverskog dijela gotovog sustava

Hardverski dio sustava se sastoji od ESP32 mikrokontrolera, 4 MCP23017 pločice za proširenje broja GPIO pinova, 3 modula sa 8 releja i jednog 5V napajanja. ESP32 i MCP23017 pločice su postavljene na dvije protoboard pločice veličine cca. 10x10 cm na dvije razine. Komponente su međusobno povezane sa easyC kablićima za komunikaciju preko I^2C protokola.



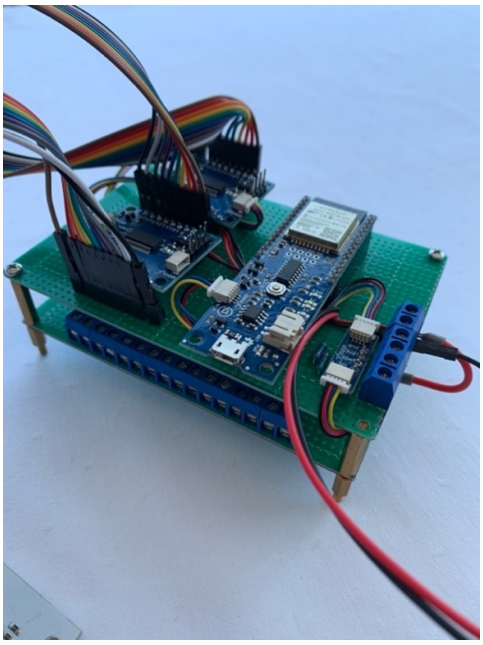
Slika 6 - Hardverski dio sustava

Dvije MCP23017 pločice služe za spajanje ulaza što znači da postoje 32 ulazna pin-a. U ovom radu su na 20 pinova spojena tipkala iz kućne instalacije, no moguće je spojiti razne senzore, npr. senzor pokreta, senzore temperature i vlažnosti i slično. Na sve ulazne pinove su spojeni pull-up otpornici od 10 kΩ.



Slika 7 - Donja razina uređaja - spajanje ulaza

Preostale dvije MCP23017 pločice služe za spajanje izlaza što daje mogućnost spajanja 32 različita izlaza. Na njih su spojena 24 releja. U ovom radu na releje je spojena rasvjeta no moguće je spojiti bilo koje trošilo jer je relej efektivno sklopka. Neke druge primjene bi bile upravljanje električnim roletama, upravljanje uređajima poput prskalica za vodu, bojlera za vodu i sličnih potrošača kod kojih jedino treba voditi računa o maksimalnoj struji releja koja iznosi 10A. Također preostalih 8 izlaza se mogu koristiti za razne namjene poput signalnih LED dioda, zujalica, upravljanje nekim drugim elektroničkim sklopom i slično.



Slika 8 - Gornja razina uređaja - ESP32 i izlazi na releje

Za napajanje sustava se koristi standardno 5V napajanje koje napaja module s relejima i ESP32, ESP32 zatim pomoću ugrađenog pretvarača napona taj napon pretvara u 3.3V i napaja sve MCP23017 pločice kako bi njihova logika odgovarala onoj na ESP32. Sve komponente su spojene na zajedničku GND sabirnicu.

Kućna elektroinstalacija je izvedena na način da su sva tipkala i žice rasvjete dovedene na jedno mjesto. Na tom mjestu je izrađena kutija od MDF-a sa izrezima kako bi se omogućilo hlađenje komponenti i napajanja koja se nalaze u kutiji. Tipkala su povezana na terminale ulaza dok su rasvjetna tijela povezana na releje. Postoji 8 običnih svijetiljka koje su spojene na releje 0-7 dok se sve ostale upravljaju putem Simply Dim načina i one su spojene po redu na ostale releje.

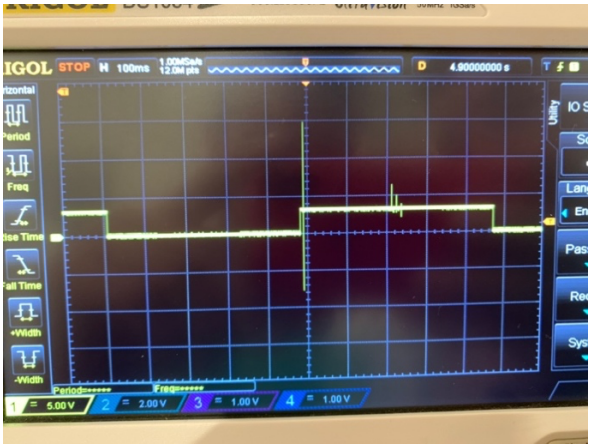
5.3. Problemi pri izradi hardverskog dijela sustava

Nakon što je sustav složen, prije spajanja ne elektroinstalaciju kuće, je testiran bez spojenih LED drivera te potom sa 2 drivera. U tim testiranjima sustav se pokazao dobro uz poneku grešku I^2C komunikacije koja se pojavila u konzoli na računalu putem serijske komunikacije. Te greške su bile rijetke i nisu utjecale na rad sustava. Nakon spajanja na elektroinstalaciju počeli su se pojavljivati problemi. Pri paljenju jednog ili više releja koji prekida fazni vodič releji bi počeli nasumično rapidno mijenjati stanje i greške u I^2C komunikaciji bi postale učestale te bi to trajalo nekoliko sekundi nakon čega bi se ulazi zamrznuli u nekom stanju i postali neresponzivni. Tek nakon resetiranja napajanja cijelog sustava bi se moglo opet koristiti, ali jednako na način kako je bilo prije reseta. Ponašanje je bilo nasumično, nije postojao uzorak u ulazima ili izlazima koji su bili pod utjecajem. Nakon detaljne analize problema uočeno je nekoliko problema.

Jedan od problema bio je to što je otpor I2C linije prema 3.3V Vcc bio prevelik. Naime ESP32 ima ugrađeni pull-up otpornik od 10 kΩ na SCL i SDA linijama koji je neophodan za rad I2C komunikacije, svaka od 4 MCP23017 pločice na sebi ima ugrađene pull-up otpornike od 10 kΩ na SCL i SDA liniji. Ukupna struja u I2C linijama mora biti manja od 3mA, idealno između 1 i 2 mA, napon je 3.3V stoga bi ukupan otpor na linijama trebao biti oko 3 kΩ. Iako je ukupan otpor u liniji iznosio oko 2 kΩ zbog otpornika u paralelnom spoju, takav način spajanja se ne smatra dobrom praksom kod spoja više uređaja stoga su pull-up otpornici na MCP23017 pločicama uklonjeni te su stavljeni otpornici od 3 kΩ koji uz ugrađene otpornike od 10 kΩ na ESP32 pločici daju ukupan otpor od 2.3 kΩ pa struja iznosi oko 1.4 mA što je zadovoljavajuće. To je poboljšalo stanje, ali nije u potpunosti riješilo problem.

Sljedeći problem koji je uočen je skok napona u žicama koje vode sa izlaznih pinova na releje. Nakon mjerenja signala pomoću osciloskopa, prilikom paljenja releja dolazi do skokova napona. Na slikama 8 i 9 se vide ti skokovi na jednom od vodića koji iznose do 15V. Iako su jako kratki, imaju veliki utjecaj na I^2C komunikaciju i rad MCP23017. Naime kada se takav napon pojavi na izlazu on može promjeniti stanje tranzistora, zato se u nekim slučajevima događalo da cijeli jedan registar MCP23017 promijeni stanje u HIGH, a nakon toga normalno dalje radi. Također takvi skokovi mogu resetirati MCP23017 nakon čega je potrebno ugasiti napajanje da bi se pločica ponovno inicijalizirala.

Slika 9 - Osciloskopsko mjerenje elektromagnetskih smetnji - šire



Slika 10 - Osciloskopsko mjerenje elektromagnetskih smetnji - uže

Postavlja se pitanje što uzrokuje te skokove napona. Pošto su releji na modulu spojeni sa zaštitom u vidu „flyback“ diode i opto-izolatora stoga su galvanski odvojeni, isključena je mogućnost da su to povratne struje sa releja. Jedina mogućnost koja preostaje su elektromagnetske smetnje koje u faznom vodiću izazivaju visoke frekvencije koje stvaraju switch-mode napajanja u radu. Rješenja za to ima više, no mnoga su komplicirana i ne garantiraju uspjeh. Nije moguće fizički razdvojiti releje od upravljačkog sklopa kako smetnje nebi imale utjecaj, mogućnost je zatvoriti upravljački sklop u Faraday-ev kavez no izrada kaveza je dosta komplicirana i ne garantira uspjeh. Najjednostavnije rješenje je staviti RLC filter između faznog i nultog vodića koji će pokupiti smetnje. Nakon ugradnje kupovnog EMI (Electro Magnetical Interference) filtera paralelno na svaki LED driver.

6. REALIZACIJA SUSTAVA – SOFTWARE

U ovom poglavlju biti će pokazan i objašnjen programski kod na mikrokontroleru te programski kod web aplikacije. Zbog velike količine i kompleksnosti koda prikazane su samo najosnovnije funkcije bitne za funkcionalnost i specifične za sustav. Cijeli kod je priložen uz rad.

Programsko rješenje mikrokontrolera je podijeljeno u dvije cijeline: dio za upravljanje relejima pomoću tipkala i dio za upravljanje putem web aplikacije. Sustav je zamišljen tako da jedno može raditi bez drugoga.

Web aplikacija je potpuno neovisna i bazira se na bazi podataka. Na njoj se izvršava početno podešavanje sustava kao i bilo kakve promjene postavki, grupa, dodavanje releja i slično. Web aplikacija je neovisna o mikrokontroleru, što znači da će ona raditi jednako u slučaju da uneseni mikrokontroler ne postoji, dok mikrokontroler može raditi bez web aplikacije, ali tek nakon što se preko nje postavi za rad.

Sustav je izrađen na način da korisnik, kada se koristi web aplikacijom, mijenja podatke u bazi podataka dok mikrokontroler periodički šalje upite nad bazom i povlači te podatke.

Web aplikacija je napravljena na način da podržava više korisnika i više uređaja. Svaki korisnik može imati više uređaja. Svaki uređaj ima svoj „token“ broj koji korisnik upisuje prilikom prvog postavljanja uređaja. Takav oblik omogućuje skalabilnost sustava i olakšava komercijalizaciju.

6.1. Kod unutar mikrokontrolera

Mikrokontroler ima dvije zadaće. Prva (primarna) zadaća mu je pratiti ulaze odnosno tipkala i kod pritiska nekog tipkala upravljati odgovarajućim relejima. Druga zadaća mu je komunikacija s web serverom, odnosno periodično slanje upita na server kako bi provjerio da li je došlo do promjene u stanjima ili konfiguraciji sustava te slanje informacije o promjeni stanja neke grupe releja.

Mikrokontroler mora konstatno provjeravati stanje ulaza u što je moguće manjim intervalima kako se ne bi dogodio pritisak tipkala koji kontroler ne registrira. Slanje HTTP zahtjeva prema web serveru i povratak podataka može trajati od nekoliko stotina milisekundi do nekoliko sekundi. Tu dolazi do problema jer dok mikrokontroler „radi“ na HTTP zahtjevu ne može čitati stanja na ulazima. Rješenje je paralelno izvođenje tih procesa na odvojenim jezgrama procesora. ESP32 ima dvojezgrenu arhitekturu procesora te je to iskorišteno na način da jedna jezgra šalje i obrađuje HTTP zahtjeve dok druga jezgra prati stanja na ulazima i upravlja relejima.

U nastavku će biti opisane osnovne funkcije koda, dok su neke pomoćne funckije izuzete. U kodu se nalaze komentari koji objašnjavaju neke specifične linije i funkcije.

Funkcija setup() :

- sadrži inicijalizacijske funkcije te dohvaća podatke iz trajne memorije.
- izvodi se samo jednom pri pokretanju mikrokontrolera

```
void setup()
{
  Serial.begin(115200);
  prefs.begin("homecontrol", false); //Inicijalizacija trajne memorije

  initWiFi();
  initMCP();
  getFromFlash(); //Dohvaćanje podataka iz trajne memorije

  //Postavljanje task-a koji izvodi funkciju serverHandler paralelno funkciji loop gdje se
serverHandler izvodi na jezgri 0, dok se loop po defaultu izvodi na jezgri 1

  xTaskCreatePinnedToCore(
      serverHandler,   /* Funkcija task-a. */
      "ServerHandler", /* Ime task-a */
      10000,           /* Velicina stacka */
      NULL,            /* Parametar */
      1,               /* Prioritet */
      &Task0,          /* Task handle */
      0);              /* Jezgra na kojoj se izvodi */

}
```

Funkcija loop() – u beskonačnoj petlji se izvodi funkcija physicalButtonsHandler()

```
void loop()
{
  physicalButtonsHandler();
}
```

Funkcija physicalButtonsHandler() objedinjuje sve funkcionalnosti potrebne za rad „fizičkog“ dijela sustava, odnosno upravljanje relejima putem tipkala. Prvo se u varijablu pressedButton sprema vrijednost funkcije inputRead() koja predstavlja trenutno stisnuto tipkalo. Zatim se pale svi releji koji odgovaraju grupi kojom upravlja stisnuto tipkalo. U ovisnosti od vrste trošila kojom relej upravlja relej se može ugasiti (Simply Dim) ili ostati upaljen/ugašen (obično trošilo). Funkcija također sprema trenutno stanje grupe u RAM memoriju te se na kraju poziva funkcija saveToFlash() koja sve parametre sprema u flash memoriju.

```
void physicalButtonsHandler()
{
  pressedButton = inputRead();
  int group = inputPin[pressedButton];
  if (pressedButton != 99)
  {
    startTime = millis();
    //Prvo se svi releji koji odgovaraju grupi koju kontrolira tipkalo postavljaju na HIGH
    for (int i = 0; i < NUM_OF_RELAYS; i++)
    {
      int relay = relayGroup[group][i];
      relayWrite(relay, HIGH);
    }
    while (buttonRead(pressedButton)) //Čekanje dok je tipkalo pritisnuto
    {
    }

    delay(200); //Delay u slučaju da je tipkalo pritisnuto prekratko

    if (millis() - startTime < 800) //Ako je tipkalo stisnuto kraće od 800 ms
    {
      if (!groupState[group]) //Ako je grupa bila ugašena
      {

        for (int i = 0; i < NUM_OF_RELAYS; i++)
        {
          int relay = relayGroup[group][i];
          if (relayType[relay])
            relayWrite(relay, LOW);
        }
        changeGroupState(group);
      }

      else if (groupState[group]) //Ako je grupa bila upaljena
      {
        for (int i = 0; i < NUM_OF_RELAYS; i++)
        {
          int relay = relayGroup[group][i];
          relayWrite(relay, LOW);
```

```
        }
        changeGroupState(group);
      }
    }

    else //Ako je tipkalo stisnuto duže od 800 ms
    {
      if (!groupState[group]) //Ako je grupa bila ugašena
      {

        for (int i = 0; i < NUM_OF_RELAYS; i++)
        {
          int relay = relayGroup[group][i];
          if (relayType[relay])
            relayWrite(relay, LOW);
        }
        changeGroupState(group);
      }

      else if (groupState[group]) //Ako je grupa bila upaljena
      {
        for (int i = 0; i < NUM_OF_RELAYS; i++)
        {
          int relay = relayGroup[group][i];
          if (relayType[relay])
            relayWrite(relay, LOW);
        }
      }
    }
  }
  saveToFlash();
  switched = group;
}
```

Funkcija serverHandler() se izvodi u beskonačnoj petlji te periodički svake sekunde šalje HTTP GET zahtjev na web server. U GET zahtjevu se nalazi token String te u slučaju da je promjenjeno stanje neke grupe uključuje se i parametar switch u kojem se nalazi broj grupe kojoj treba promijeniti stanje u bazi podataka. Nakon slanja zahtjeva te ako je server odgovorio sa kodom 200 dohvaća se sadržaj odgovora koji je JSON datoteka te se poziva funkcija saveFromJson() koja dekodira JSON datoteku i njen sadržaj sprema u memoriju. Zatim se preuzete vrijednosti uspoređuju sa prethodnim vrijednostima te se mjenja stanje grupa rasvjete kod kojih je došlo do promjene odnosno koje je korisnik promijenio na web aplikaciji. U slučaju greške ili gubitka veze sa internetom funkcija se neće pravilno izvesti, ali pošto se jedina izvodi na jezgri procesora neće utjecati na rad te će, kada bude u mogućnosti, početi opet slati zahtjeve bez posljedica na rad sustava.

```
void serverHandler(void *pvParameters)

{
  while (true)
  {
    String payload = "{}";
    if ((WiFi.status() == WL_CONNECTED))
    {
      if (switched != 99)
      {
        http.begin(serverName + "&switch=" + String(switched));
      }
```

```
      else
      {
        http.begin(serverName);
      }
      int httpCode = http.GET();
      if (httpCode == 200)
      {
        switched = 99;
        payload = http.getString();
        bool temp[NUM_OF_GROUPS];
        for (int i = 0; i < NUM_OF_GROUPS; i++)
        {
          temp[i] = groupState[i];
        }
        saveFromJson(payload);
        for (int i = 0; i < NUM_OF_GROUPS; i++)
        {
          if (temp[i] != groupState[i])
          {
            switchGroup(i);
          }
        }
        saveToFlash();
      }


 else
      {
        Serial.println("Error on HTTP request");
      }
      http.end();
    }
    vTaskDelay(1000);
  }
}
```

6.2. Web aplikacija

Web aplikacija je izrađena u jeziku Python sa frameworkom Django prema MVT (Model View Template) arhitekturi. Model predstavlja bazu podataka, View sadrži funkcije koje obrađuju zahtjeve na strani servera te prosljeđuju podatke u Template koji predstavlja HTML datoteke koje se prikazuju korisniku. U ovom poglavlju biti će opisane samo funkcije u View-u i Model jer Template nije toliko relevantan za prikaz strukture aplikacije. Cijela aplikacija je priložena uz rad.

Model:

- klase u datoteci models.py predstavljaju tablice u bazi podataka. Atributi klasa predstavljaju stupce u tim tablicama čija se svojstva definiraju pozivanjem raznih funkcija
- baza podataka aplikacije je vrlo jednostavna i sastoji se od 4 tablice: Board, RelayGroup, Input, Relay
- baza je koncipirana u skladu sa kodom na samom mikrokontroleru kako bi se olakšao prijenos podataka između mikrokontrolera i baze

Klasa Board predstavlja ESP32 pločicu. Ona ima svoje ime (*name*), *token* koji predstavlja jedinstveni broj svake pločice i ujedno je i primarni ključ ove tablice, te svaka pločica pripada jednom ili više korisnika, dok korisnik može imati više pločica stoga su tablica User i Board povezane vezom više-prema-više preko atributa *user*.

```
class Board(models.Model):
  name = models.CharField(max_length=300)
  token = models.CharField(max_length=10,primary_key=True)
  user = models.ManyToManyField(User,related_name='board')
  def __str__(self):
      return self.name
```

Klasa RelayGroup služi za definiranje grupa koje pripadaju određenoj pločici. Atribut *group_id* odgovara vrijednosti između 0 i 23 kako bi odgovarao kodu na mikrokontroleru koji podržava maksimalno 24 grupe. Svaka grupa ima ime (*name*). Također se sprema trenutno stanje grupe u atributu *state*. Grupa je povezana sa pločicom preko varijable *token* koja je primarni ključ.

```
class RelayGroup(models.Model):

  group_id = models.IntegerField(validators=[MaxValueValidator(24),MinValueValidator(0)])
  name = models.CharField(max_length=300)
  state = models.BooleanField(default=False)
  token = models.ForeignKey(Board,on_delete=models.CASCADE)
  def __str__(self):
      return self.name
```

Klasa Input predstavlja ulazne pinove. Atribut *board* povezuje tablicu s pločicom *Board* te je on vanjski ključ. Svaki ulaz ima ime *name* te *inputPin_id* koji predstavlja broj između 0 i 31 pošto na mikrokontroleru postoji 32 ulazna pina. Atribut *groupToControl* je također vanjski ključ tablice koji povezuje tablicu s grupom *RelayGroup* kojom ulazni pin upravlja.

```
class Input(models.Model):
  board = models.ForeignKey(Board,on_delete=models.CASCADE)
  name = models.CharField(max_length=300)
  inputPin_id = models.IntegerField(validators=[MaxValueValidator(31),MinValueValidator(0)])
  groupToControl = models.ForeignKey(RelayGroup,on_delete=models.SET_NULL, default=None,
blank=True,null=True)
  def __str__(self):
      return self.name
```

Klasa Relay predstavlja izlaze, odnosno releje. Svaki ima svoje ime *name, token* koji predstavlja vezu na pločicu te je vanjski ključ tablice, atribut *type* koji označava da li je trošilo na releju obično (False) ili Simply Dim (true), atribut *group* koji je vanjski ključ na tablicu *RelayGroup* te označava grupu kojoj relej pripada te atribut *pin* koji označava na kojem se pinu na pločici nalazi relej. Na pločici postoje 32 izlazna pina stoga atribut *pin* može imati vrijednost između 0 i 31.

```
class Relay(models.Model):
  name = models.CharField(max_length=300)
  token = models.ForeignKey(Board,on_delete=models.CASCADE)
  type = models.BooleanField(default=False)
  group = models.ForeignKey(RelayGroup,on_delete=models.CASCADE,blank=True,null=True)
  pin = models.IntegerField(validators=[MaxValueValidator(31),MinValueValidator(0)])
  def __str__(self):
      return self.name
```

View:

- funkcije u datoteci views.py se pozivaju kod pristupa pripadajućem URL-u definiranom u datoteci urls.py
- funkcije vrše upite nad bazom podataka te obrađuju podatke
- nakon izvršenja funkcije korisniku se prikazuje odgovarajuća HTML datoteka kojoj se prosljeđuje određeni skup podataka definiran u context = {} čiji se sadržaj dalje manipulira direktno u HTML datoteci putem Django template jezika

Sadržaj urls.py datoteke koji definira koji view se poziva kod pristupa određenom URL putu:

```
urlpatterns = [
  path('',views.dashboard,name="dashboard"),
  path('settings/',views.settings,name="settings"),
  path('analytics/',views.analytics,name="analytics"),
  path('users/',views.users,name="users"),
  path('register/',views.registerPage,name="register"),
  path('login/',views.loginPage,name="login"),
  path('logout/',views.logoutUser,name="logout"),
  path('update/',views.update,name="update"),
  path('initialSetup/',views.initialSetup,name="initialSetup")
]
```

loginPage funkcija služi za autentifikaciju korisnika i preusmjeravanje na Dashboard ako je korisnik uspješno logiran

```
def loginPage(request):
    if request.user.is_authenticated:
        return redirect('dashboard')
    else:
        if request.method == 'POST':
            username = request.POST.get('username')
            password = request.POST.get('password')

            user = authenticate(request, username=username, password=password)
            if user is not None:
                login(request, user)
                return redirect('dashboard')
            else:
                messages.info(
                    request, 'Korisničko ime ili lozinka ne postoji!')
        context = {}
        return render(request, 'login.html', context)
```

logOut funkcija je trivijalna funkcija koja služi za odjavu korisnika

```
def logoutUser(request):
    logout(request)
    return redirect('login')
```

dashboard funkcija dohvaća sve grupe koje pripadaju svim pločicama koje ulogirani korisnik ima te prikazuje dashboard.html kojem prosljeđuje QuerySet sa svim korisnikovim grupama. Ako korisnik nema definiranu nijednu grupu biti će preusmjeren na postavke. Iznad funkcije se nalazi decorator @login_required koji zahtjeva da je korisnik ulogiran kako bi se moglo pristupiti stranici, ako neautorizirani korisnik pokuša pristupiti biti će preusmjeren na login stranicu.

```
@login_required(login_url='login')
def dashboard(request):
    current_user = request.user
    boards = Board.objects.filter(user=current_user)
    groups = RelayGroup.objects.none()
    for board in boards:
        groups = groups | RelayGroup.objects.filter(token=board.token)

    if not(groups.exists()):
        return redirect('settings')
    context = {'groups': groups}
    return render(request, 'dashboard.html', context)
```

initialSetup funkcija služi za početno postavljanje sustava kada korisnik nema definiranu nijednu pločicu ili za postavljanje nove pločice. Funkcija čita podatke iz POST-a. Prvi podatak je *boardToken* koji predstavlja token nove pločice, ako uneseni token odgovara postojećoj pločici zahtjev je preusmjeren na Dashboard, inače se čitaju podaci o novoj pločici i spremaju u bazu podataka. Na kraju se prikazuje ponovno initialSetup.html no ako je poziv nakon što se unesu svi podaci o pločici u context se prosljeđuje i boardToken koji kod toga poziva odgovara pločici koja se nalazi u bazi stoga se korisnik preusmjerava na Dashboard.

```
def initialSetup(request):
    if request.method == 'POST':
        boardToken = request.POST.get('boardToken')
        if boardToken is not None:
            try:
                board = Board.objects.get(token=boardToken)
            except Board.DoesNotExist:
                board = None

            if board:
                context = {}
                return redirect('/')
            elif request.POST.get('boardName'):
                user = User.objects.get(username = request.user.username)
                boardName = request.POST.get('boardName')
                boardToken = request.POST.get('boardToken')
                b = Board(name=boardName, token=boardToken)
                b.save()
                user.board.add(b)

                for i in range(0,int(request.POST.get('numOfRelays'))):
                    relayName=request.POST.get('relayName'+str(i))
                    if request.POST.get('relayType'+str(i))=="on":
                        relayType=True
                    else:
                        relayType=False
                    r =
Relay(name=relayName,token=Board.objects.get(token=boardToken),type=relayType,pin=i)
                    r.save()
                context = {}
                return redirect('/')
            else:
                context = {'boardInitialized': 1, 'boardToken': boardToken}
                return render(request, 'initialSetup.html', context)

    context = {'boardInitialized': 0}
    return render(request, 'initialSetup.html', context)
```

settings funkcija primarno prikazuje settings.html kojem u context prosljeđuje QuerySet-ove relays, boards, groups i inputs. Kod popunavanja neke od formi koje se nalaze u settings.html se POST zahtjevom šalju podaci koje funkcija čita te sprema u bazu podataka.

```
@login_required(login_url='login')
def settings(request):
    user = request.user
    boards = Board.objects.filter(user=user)
    relays = Relay.objects.all()
    groups = RelayGroup.objects.all()
    inputs = Input.objects.all()

    if(request.method == 'POST'):
```

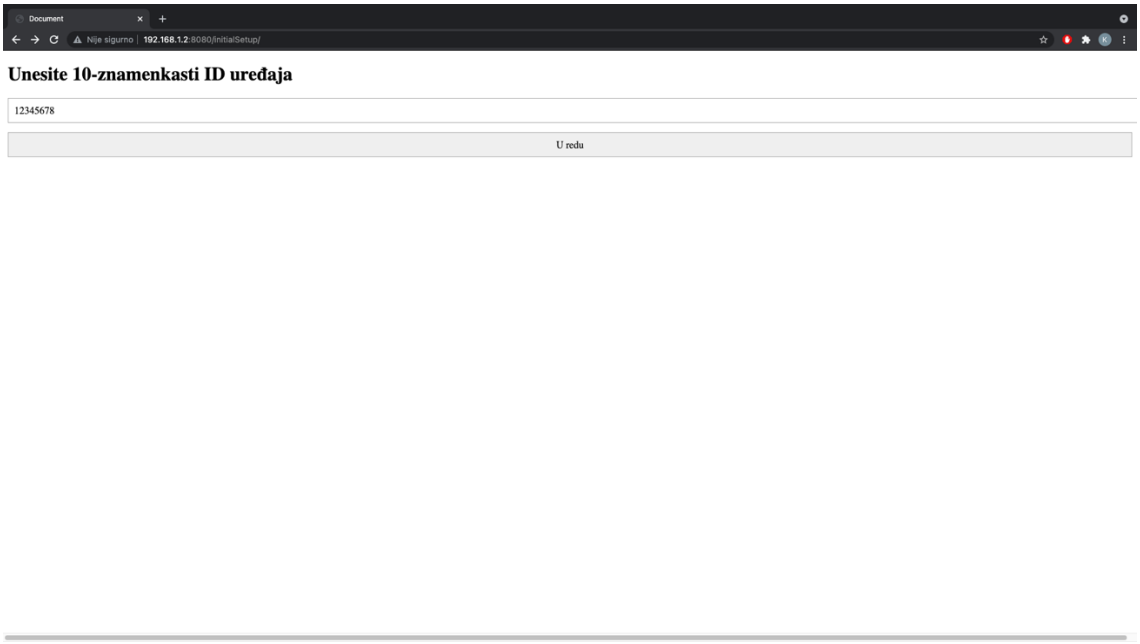
```
if (request.POST.get('newInputName')):
            newName = request.POST.get('newInputName')
            newPin = request.POST.get('newInputPin')

            i = Input(name=newName,inputPin_id=newPin,
                      board=Board.objects.get(token=request.POST.get('boardToken')))
            i.save()
            return HttpResponseRedirect(request.path)

        if (request.POST.get('newGroup')):
            newName = request.POST.get('newGroup')
            g = RelayGroup(name=newName,group_id=groups.count(),
                           token=Board.objects.get(token=request.POST.get('boardToken')))
            g.save()
            return HttpResponseRedirect(request.path)


        for input in inputs:
            inputName=request.POST.get('inputName.'+str(input.id))
            inputPin = request.POST.get('inputPin.'+str(input.id))
            inputGroup = request.POST.get('inputGroup.'+str(input.id))
            boardToken = request.POST.get('boardToken')
            if(input.board.token == boardToken):
                if inputGroup:
                    group = RelayGroup.objects.get(id=inputGroup)
                else:
                    group = None
                if(inputName==''):
                    Input.objects.filter(id=input.id).delete()
                else:
                    i = Input(id=input.id,name=inputName,inputPin_id=inputPin,
                              groupToControl=group,board=Board.objects.get(token=boardToken))
                    i.save()

        count = 0
        for group in groups:
            groupName = request.POST.get('group.'+str(group.id))
            boardToken = request.POST.get('boardToken')
            if(group.token.token == boardToken):
                if(groupName==''):
                    RelayGroup.objects.filter(id=group.id).delete()
                else:
                    g = RelayGroup(id=group.id,group_id=count,name=groupName,
                                   token=Board.objects.get(token=boardToken))
                    g.save()
                    count+=1

        for relay in relays:
            nameNew = request.POST.get('relayName.'+str(relay.id))
            groupNew = request.POST.get('relayGroup.'+str(relay.id))
            if request.POST.get('type.'+str(relay.id)):
                typeNew=True
            else:
                typeNew=False

            if groupNew:
                group = RelayGroup.objects.get(id=groupNew)
            else:
                group = None

            boardToken = request.POST.get('boardToken')
            if(relay.token.token == boardToken):
                 r = Relay(id = relay.id, name = nameNew, group = group,
                token=Board.objects.get(token=boardToken),type= typeNew, pin = relay.pin)
                 r.save()
        return HttpResponseRedirect(request.path)

    context = {'relays':relays, 'boards':boards, 'groups':groups, 'inputs':inputs}
    return render(request, 'settings.html', context)
```

update funkcija primarno služi da bi mikrokontroler mogao poslati GET upit te tako promijeniti stanje grupe u bazi. Funkcija vraća JSON datoteku koja sadrži ažurne postavke sustava kako bi mikrokontroler bio sinkroniziran sa bazom. Funkcija se također može koristiti kako bi se stanje određene grupe moglo jednostavno promijeniti putem nekog vanjskog uređaja bez pristupa web aplikaciji (npr. glasovni asistent na mobilnom uređaju se programira da se nakon neke glasovne komande pošalje GET upit na adresu servera koji u GET parametrima sadrži token i grupu kojom želimo upravljati. Time se na jednostavan način dobije glasovno upravljanje rasvjetom)

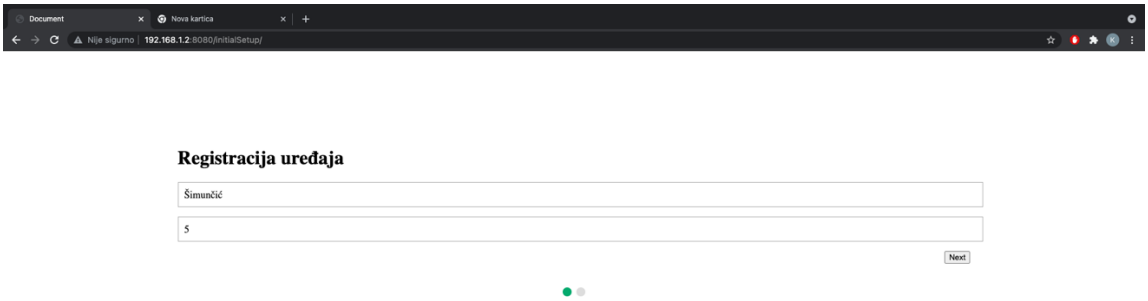
```
def update(request):
    boardToken = request.GET.get("token")
    groupToSwitch = request.GET.get("switch")
    if(groupToSwitch):
        group = RelayGroup.objects.filter(token=boardToken).get(group_id = groupToSwitch)
        s = RelayGroup.objects.get(id = group.id)
        s.state = not(s.state)
        s.save()
    if(request.GET.get("redirect")):
        return redirect('/')
    data=list()
    data.append(list(RelayGroup.objects.filter(token=boardToken).values('group_id','state')))

data.append(list(Relay.objects.filter(token=boardToken).values('pin','group__group_id','type')
))

data.append(list(Input.objects.filter(board=Board.objects.get(token=boardToken)).values('input
Pin_id','groupToControl__group_id')))
    return JsonResponse(data,safe=False)
```

6.3. Web aplikacija – slike zaslona

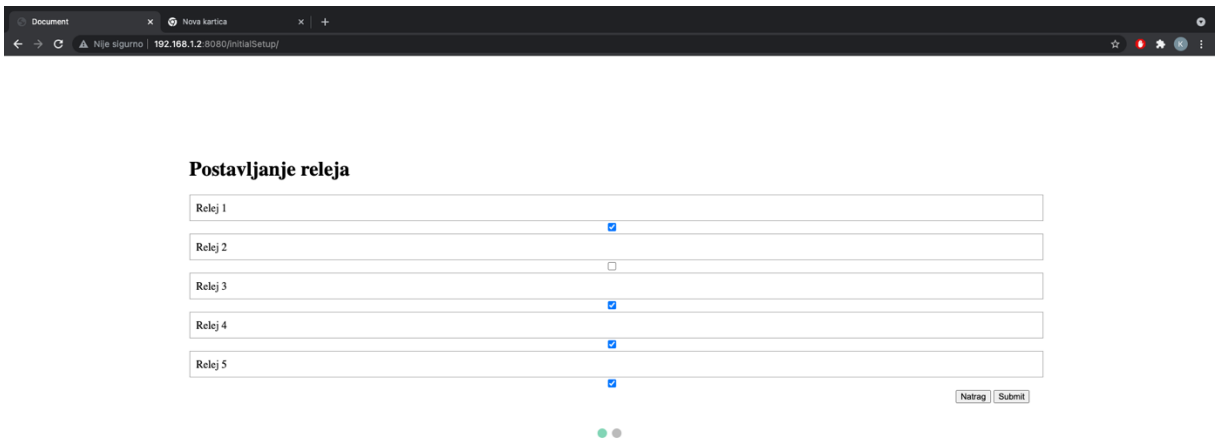
U ovom potpoglavlju su prikazane slike zaslona odnosno izgled web aplikacije. Slike su poredane kronološki kako se koristi sustav: korisnik prvo upisuje ID uređaja odnosno pločice, zatim broj releja i njihove opise, kada je završeno početno postavljanje korisnik je preusmjeren na stranicu „Postavke“, sa glavne stranice, odnosno „Dashboard“ se može upravljati postavljenim grupama rasvjete.



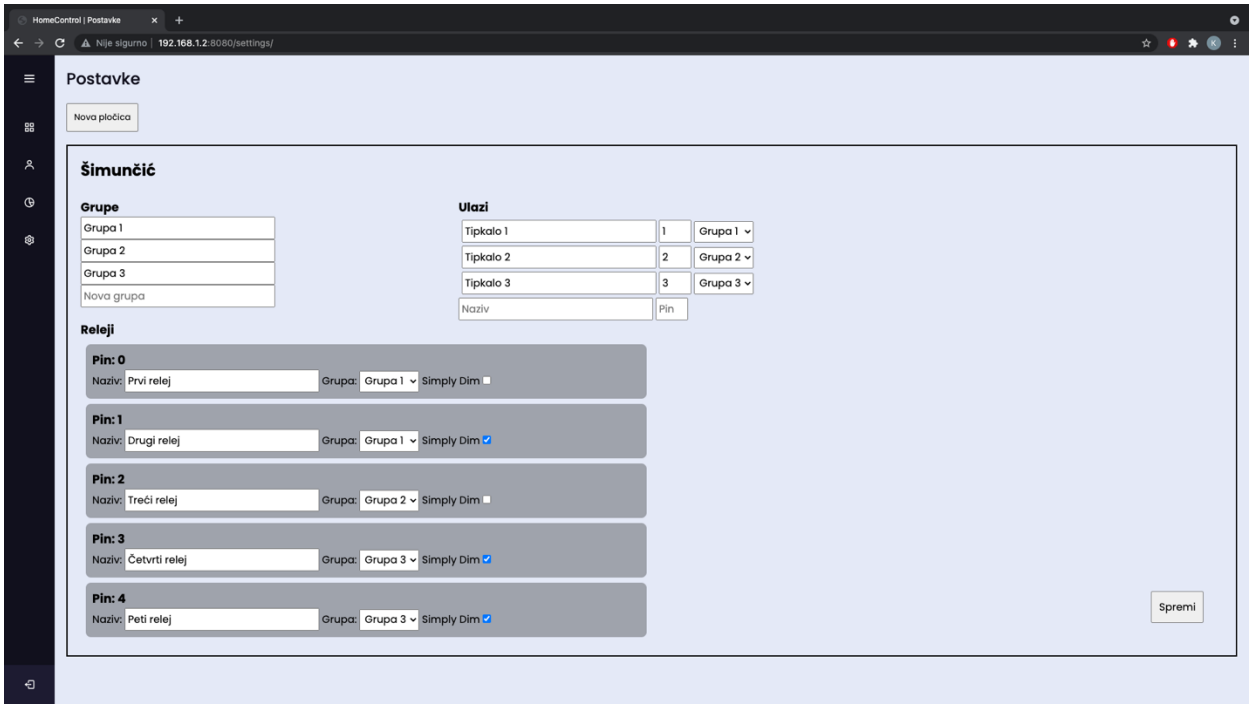
Slika 11 - Početno postavljanje - unos ID broja



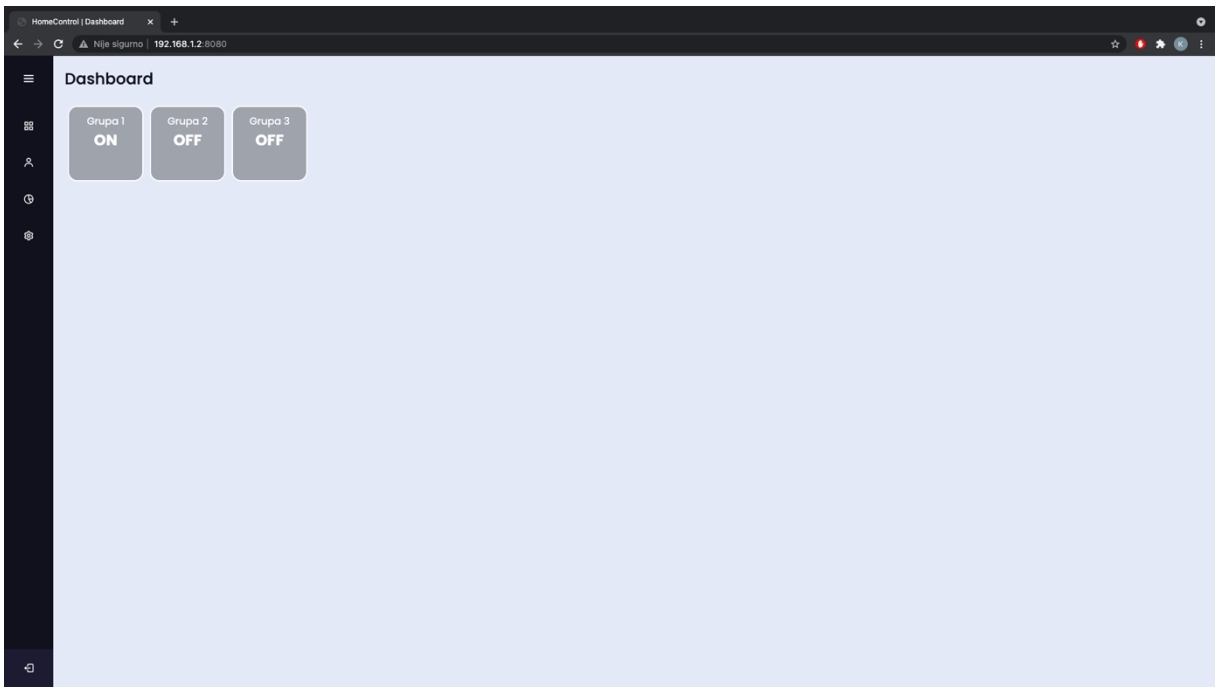
Slika 12 - Početno postavljanje - unos naziva i broja releja



Slika 13 - Početno postavljanje - unos releja



Slika 14 - Postavke



Slika 15 - Dashboard - upravljanje grupama releja

7. MOGUĆNOSTI ZA POBOLJŠANJE SUSTAVA

Sustav je izrađen na način da je moguće jednostavno proširiti njegove mogućnosti i da ga je moguće skalirati. U ovom poglavlju biti će opisane neke funkcionalnosti koje se mogu nadodati, a nisu u trenutnom sustavu zbog nedostatka vremena ili kompleksnosti izvedbe.

Analitika - u web aplikaciji se može implementirati sekcija „Analitika“ koja bi sadržavala grafikone koji prikazuju potrošnju energije, vrijeme korištenja rasvjete ili ostalih uređaja i sl. To se može jednostavno implementirati na način da se u bazi podataka tablici Relay pridoda polje koje bi sadržavalo snagu uređaja spojenog na određeni relej koje bi korisnik morao ispuniti prilikom inicijalizacije releja te da se postavi tablica koja bi na neki način spremala vremenske pečate promjene stanja releja. Algoritmom bi se odredilo ukupna potrošnja energije korz dan, tjedan ili mjesec te bi se podaci mogli grafički prikazati korisniku. To bi bilo korisno da korisnik može vidjeti koji su mu najveći potrošači energije te uskladiti svoje navike u odnosu na to kako bi uštedio električnu energiju. Također se mogu implementirati i algoritmi strojnog učenja ili umjetne inteligencije koji bi učili navike korisnika, nudili mu preporuke kako i gdje uštediti energiju ili čak samostalno upravljali rasvjetom i uređajima u skladu s korisnikovim navikama u cilju smanjenja potrošnje električne energije.

Optimizacija koda u svrhu skalabilnosti – sustav je rađen prvobitno za konkretne potrebe jedne obiteljske kuće, iako nije plan proširiti ili komercijalizirati sustav neke stvari su ipak napravljene na način da je to moguće napraviti poput baze podataka bazirane na korisnicima i slično. Dok je ostavljena ta mogućnost ipak je sustav rađen za jednog korisnika te se nije previše obraćala pažnja na optimizaciju funkcija odnosno spremanju podataka i zaprimanju zahtjeva od većeg broja korisnika i uređaja. Kod na mikrokontroleru također nije maksimalno optimiziran no on u svakom slučaju radi istu stvar stoga njegova optimizacija nije neophodna ali je preporučena, dok kod na web aplikaciji može predstavljati problem u slučaju velikog broja korisnika i bilo bi ga nužno optimizirati prije komercijalizacije sustava.

Sigurnost – u sustavu nije posvećena pažnja sigurnosti protiv cyber napada iz razloga što se u konkretnom slučaju koristi na privatnoj mreži te upravlja samo rasvjetom što čak i da dođe do neovlaštenog upravljanja ne predstavlja veliki problem. U slučaju komercijalizacije sustava ili upravljanje uređajima čija neovlaštena kontrola predstavlja sigurnosni rizik poput garažnih vrata ili elektronske brave bilo bi potrebno implementirati enkripciju komunikacije između mikrokontrolera i servera uz što više metoda za zaštitu servera protiv neovlaštenih pristupa.

Redizajn hardvera – hardverski dio sustava je napravljen u obliku prototipa. Takav oblik je izvrstan za brze promjene, testiranje i jeftinu izradu sustava no ne nudi pouzdanost i robusnost koja se od njega očekuje. Nakon ekstenzivnog testiranja u svim mogućim uvjetima rada te rješavanja svih potencijalnih problema biti će potrebno izraditi prilagođenu tiskanu pločicu na koju bi se postavile i zalemile sve komponente sustava. Pošto su veliki problem sustavu predstavljale elektromagnetske smetnje bilo bi potrebno implementirati što je moguće više načina zaštite protiv istih i sličnih smetnja koje bi mogle naškoditi radu sustava. Tiskana pločica se treba dizajnirati tako da bude što manjih dimenzija, ali paziti na raspored i izvedbu kako ne bi došlo do neželjenih pojava poput pregrijavanja i slično.

UX i UI dizajn – pažnja nije posvećena dizajnu korisničkog sučelja na web aplikaciji. Moguće je urediti CSS i JavaScript datoteke kako bi dizajn sučelja postao ugodan i moderan. Također je na isti način moguće poboljšati iskustvo korisnika na način da se raspored sadržaja prilagodi kako bi bilo jasnije što je što te kako bi korisnik mogao brzo i jednostavno pronaći ono što treba. U slučaju komercijalizacije sustava bi bilo potrebno napisati vodiće i upute za korištenje sustava.

Podrška za DALI protokol – iako sustav radi na ovaj način on je tako napravljen iz nužde. Elegantniji način kontrole rasvjetom je putem DALI protokola koji bi omogućio daljinsko upravljanje intezitetom i bojom rasvjete, slanje upita o stanju prema rasvjeti što bi omogućilo puno jednostavniji kod koji bi bio brži i točniji od sadašnjeg. Eliminirali bi se mogući problemi koji mogu nastati u slučaju da mikrokontroler krivo procjeni stanje grupe rasvjete ili problemi koji nastaju u slučaju da „dimmabilna“ rasvjeta izgubi sinkronizaciju sa ostalom takvom rasvjetom u grupi. Također bi se spajanje rasvjete puno pojednostavnilo jer bi se DALI kontrolne žice spajale na jednu sabirnicu i ujedno bi sustav bio nečujan jer se nebi čuo „klik“ releja prilikom svake promjene stanja.

Automatizacija – na web aplikaciji je moguće implementirati sučelje u kojem bi korisnik mogao raditi skripte automatizacije. Elementi skripte bi bili ulazi, izlazi, vremenski pečati i uvjeti. Na taj način bi korisnik mogao jednostavno napraviti automatizacije poput „ako je [22:00 sati] upali [grupa 6]“ i „ako je [ulaz 3] == 1 upali [izlaz 1]“ gdje bi posljednja automatizacija mogla predstavljati jednostavan alarmni sustav ukoliko je na ulaz 3 spojen senzor pokreta, a na izlaz 1 sirena. Skripte bi se radile u jednostavnom grafičkom sučelju kako bi bilo intuitivnije za korisnike koji nemaju iskustva u programiranju. Također bi postojale standardne skripte koje bi radile česte automatizacije gdje bi korisnik morao samo definirati željene ulaze i izlaze.

8. ZAKLJUČAK

Cilj rada je bio izraditi sustav prema relativno jednostavnim zahtjevima vlasnika obiteljske kuće. To je uspješno realizirano te je sustav još dodatno proširen sa funkcionalnostima koje nisu zahtjevane. Sustav će se nastaviti dalje razvijati prema mogućnostima koje su navedene u poglavlju 6.

Iako sustav u trenutku pisanja ovog rada funkcionira kako treba, njegova pouzdanost će se tek pokazati kroz svakodnevno korištenje dugi niz godina koje slijedi.

Kroz izradu sustava i sve probleme na koje sam naišao naučio sam jako puno o radu mikrokontrolera, njegovom programiranju, specifičnim problemima i njihovim rješenjima. Također sam puno naučio o razvoju web aplikacija pomoću Django frameworka koji je trenutno jedno od vodećih tehnoloških rješenja za razvoj web aplikacija u svijetu.

LITERATURA

1. Wikipedia – HTML
   https://hr.wikipedia.org/wiki/HTML
   pristupljeno 30.8.2021.

2. Službena PlatformIO dokumentacija
   https://docs.platformio.org/
   pristupljeno 13.9..2021.

3. Službena Django dokumentacija
   https://docs.djangoproject.com/
   pristupljeno 13.9.2021.

4. RandomNerdTutorials
   https://randomnerdtutorials.com/
   pristupljeno 13.9.2021.

5. ESP32 technical reference manual
   https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf
   pristupljeno 15.8.2021.

6. Službena dokumentacija ArduinoJSON biblioteke
   https://arduinojson.org/v6/doc/
   pristupljeno 1.9.2021.

7. Službena Arduino dokumentacija
   https://www.arduino.cc/en/main/docs
   pristupljeno 1.9.2021.

8. ESP32 Wroom 32D Datasheet
   https://e-radionica.com/productdata/ESP32-Wroom-32D.pdf
   pristupljeno 15.8.2021.

9. MCP23017 Datasheet
   https://ww1.microchip.com/downloads/en/devicedoc/20001952c.pdf
   pristupljeno 15.8.2021.

10. Understanding the I^2C bus, Jonathan Valdez, Jared Becker
    https://www.ti.com/lit/an/slva704/slva704.pdf?ts=1631886575763&ref_url=https%253A%252F%252Fwww.google.com%252F
    pristupljeno 19.8.2021.

11. Službene stranice DALI-alliance
    https://www.dali-alliance.org/dali/ , pristupljeno 20.8.2021.

POPIS SLIKA

POPIS PRILOGA

Uz rad se prilaže main.cpp datoteka koja predstavlja kod na mikrokontroleru i folder HomeControlWeb u kojem se nalazi cijelokupni Django projekt web aplikacije.