Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Alen Slejko

Brezžično vozlišče za pametno hišo

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: viš. pred. dr. Robert Rozman

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Avtomatizacija obstoječih objektov je zahtevnejši problem, ker so posegi v obstoječe napeljave bistveno težji in dražji. Kljub temu napredek brezžičnih tehnologij omogoča zasnovo lokalnih brezžičnih omrežij, ki lahko povežejo različne naprave in tipala v lokalno omrežje brez večjih posegov v obstoječe napeljave. Prav tako tehnologija računalniških sistemov ponuja vse več cenovno dostopnih in tehnološko naprednih naprav oziroma sistemov. Izdelajte prototip takšnega cenovno dostopnega sistema s pomočjo naprav, ki jih najdete na trgu. Dopolnite sistem z lastnimi rešitvami tam, kjer je to potrebno in smiselno. Sistem naj zagotovi uporabniku udobno, prijazno in cenovno dostopno avtomatizacijo posameznega bivalnega prostora. Ob tem zagotovite tudi mrežno povezljivost in enostavno upravljanje sistema tudi na daljavo.

Zahvaljujem se viš. pred. dr. Robertu Rozmanu za napotke in pomoč pri izdelavi diplomskega dela. Zahvala gre tudi Matjažu Štrancarju za vso pomoč pri izdelavi tiskanega vezja vozlišča. Zahvaljujem se tudi svoji družini, dekletu in prijateljem, ki so me spremljali in spodbujali skozi celoten študij.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
USB	universal serial bus	univerzalno serijsko vodilo
IOT	internet of things	internet stvari
MQTT	message queuing telemetry transport	protokol za prenos telemetrijskih sporočil
WEP	wired equivalent privacy	zasebnost enaka žični povezavi
WPA	wireless protected access	zaščiten brezžični dostop
TCP	transmission control protocol	protokol za nadzor prenosa
UDP	user datagram protocol	nepovezovalni protokol za prenašanje paketov
SPI	serial peripheral interface	serijski periferni vmesnik
GPIO	general purpose input/output	splošnonamenski vhod-izhod
JSON	JavaScript object notation	notacija objekta JavaScript
MAC	media access control	nadzor dostopa do povezave
IP	internet protocol	internetni protokol

Povzetek

Naslov: Brezžično vozlišče za pametno hišo

Avtor: Alen Slejko

Avtomatizacija obstoječe hiše je lahko zahtevna s stališča vgrajevanja tipal in sprememb v obstoječi napeljavi. To težavo lahko rešimo z uporabo brezžičnih vozlišč, ki jih namestimo v vsak prostor posebej in tako na enostaven ter učinkovit način pridobimo vpogled in nadzor nad celotnim objektom. Cilj diplomske naloge je celostni razvoj vozlišča, ki komunicira z brezžičnim omrežjem Wi-Fi. Izdelano je bilo vezje, ki temelji na procesnem modulu ESP8266, sprogramiranem v jeziku C++. Vozlišče za komunikacijo z osrednjim sistemom uporablja protokol Mqtt. Osrednji sistem je implementiran s pomočjo programskega dodatka Home Assistant na računalniku Raspberry PI. Vozlišče omogoča zaznavanje temperature, vlage, gibanja, osvetljenosti in krmiljenje običajne hišne naprave pri napetosti 220 V. Poleg vezja je bila izdelana tudi programska oprema, ki poskrbi za celotno delovanje in nastavitve ob zagonu. Za komunikacijo z uporabnikom je bil izdelan preprost in pregleden uporabniški vmesnik, v katerem lahko uporabnik upravlja sistem in spreminja njegove nastavitve. V zaključku je prikazano še praktično delovanje vozlišča z izvedbo vseh potrebnih nastavitev ter povezave z osrednjim sistemom.

Ključne besede: Multisenzor, ESP8266, brezžično vozlišče, brezžično omrežje, pametna hiša.

Abstract

Title: Wireless Node for a Smart Home

Author: Alen Slejko

The transformation of an old house into a smart home can be time- and money-consuming, especially due to the integration of new sensors and modifications to the existing electrical wiring. However, this problem can be easily and effectively solved by installing wireless node devices in separate areas to enable monitoring and control of the whole building. The aim of this thesis is to develop a multisensor node connected to Wi-Fi network. The result of presented work is an electronic circuit board functioning as a wireless node. The main part is an ESP8266 microcontroller module programmed in C++ language. The node is connected to the central system using the Mqtt protocol. The central part of the system is implemented using program Home Assistant running on Raspberry PI computer. The node features the ability to measure temperature, humidity, movement, light level, and the ability to control a common household 220 V device. In addition to the hardware, the complete software was developed to support a complete node operation and setup procedure. Communication between the user and the node is carried through a user-friendly interface that allows the user to easily control the node's operation and settings. Lastly, a complete practical case of the node's integration in the home automation system is presented.

Keywords: Multisensor, ESP8266, wireless node, wireless network, smart home.

Poglavje 1

Uvod

Pojem pametne hiše sega že v zgodovino, vendar z zelo drugačno interpretacijo kot jo imamo sedaj ob misli na ta izraz. Kljub temu v osnovi izraz pomeni isto – skupek povezanih naprav, ki ljudem olajšajo vsakdanja opravila. Med leti 1901 in 1920 se je začel razvoj naprav, ki so ljudem lajšale dela v hiši [20]. Naprave kot so sesalec, hladilnik, sušilnik perila in likalnik so v tistem času predstavljali ogromen napredek pri hišnih opravilih. Žal te naprave med seboj niso bile povezane, zato je človek moral rokovati z vsako posebej. Leta 1966 pa je zaživel prvi sistem pametne hiše v današnjem pomenu besede, imenovan ECHO (angl. Electronic Computer Home Operator) [22]. Razvil ga je Jim Sutherland in prvo verzijo postavil v svoji hiši. Sistem je omogočal krmiljenje luči, vklop in izklop naprav v hiši ter celo stroškovni preračun seznama nakupovanja. Žal zaradi svoje velikosti sistem nikoli ni zaživel, dal pa je dobro iztočnico za razvoj sodobnejših pametnih sistemov. Dandanes so se komponente zelo zmanjšale in sistem pametne hiše lahko upravljamo s pomočjo računalnika v velikosti bančne kartice.

Načrtovanje pametne hiše ponavadi začnemo že pri gradnji novega objekta. Težava nastane pri že obstoječih objektih, saj je potrebno celotno električno napeljavo spremeniti in to je lahko stroškovno zelo potratno. To težavo lahko rešimo z uporabo brezžičnih vozlišč, ki jih namestimo nad omet vsakega prostora in tako pridobimo podatke o celotni hiši. Sistem za nadzor nad lučmi

pa lahko vgradimo v kabelske razvodnice. Komponente današnjih krmilnikov so namreč zelo majhne, za njihovo delovanje pa potrebujemo le vir električne energije. Vse te naprave, ki so razdeljene po hiši in povezane v omrežje, imenujemo tudi internet stvari - IOT (angl. Internet of things).

V diplomskem delu je predstavljena izdelava cenovno ugodnega vozlišča za uporabo v obstoječih objektih, ki jim želimo dodati funkcionalnost pametne hiše. Glavni mikrokrmilnik je modul ESP8266, ki za komunikacijo uporablja brezžično tehnologijo Wi-Fi. Vozlišče je zasnovano za preprosto namestitev v prostoru za merjenje temperature, vlage, gibanja in osvetljenosti; ima pa tudi možnost krmiljenja ene zunanje naprave pri standardni napetosti 220 V. Za nadzor nad vozliščem je uporabljen odprtokodni program Home Assistant.

V nadaljevanju tega dela so v 2. poglavju najprej opisane uporabljene komponente za sestavo vozlišča, v 3. poglavju sledi prikaz načrtovanja izdelave in testiranja vezja. V 4. poglavju se nahaja opis in pregled protokola za komunikacijo ter vzpostavitve osrednjega avtomatizacijskega sistema. Zatem je v 5. poglavju opisan celotni postopkek izdelave programske opreme, v predzadnjem poglavju se nahaja še praktičen prikaz delovanja z vsemi potrebnimi postopki nastavitev in povezave v sistem. Delo zaključujejo še sklepne ugotovitve.

Poglavje 2

Opis uporabljenih komponent

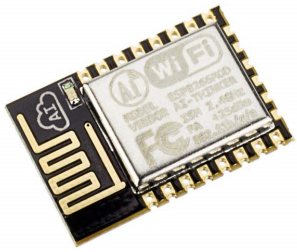
Celoten sistem hišne avtomatizacije, predstavljene v tej diplomski nalogi, je sestavljen iz dveh glavnih gradnikov: brezžičnega vozlišča in osrednjega sistema za nadzor. Na vozlišču je mikrokrmilnik ESP8266, ki skrbi za komunikacijo z osrednjim sistemom, bere vrednosti iz vhodnih in upravlja z izhodnimi napravami. Osrednji sistem je realiziran z računalnikom Raspberry Pi, ki ima nameščeno programsko opremo za hišno avtomatizacijo.

2.1 Mikrokrmilnik ESP8266

ESP8266 je zelo majhen modul v velikosti kovanca, ki podpira brezžično povezavo Wi-Fi [23]. Modul je razvilo kitajsko podjetje Espressif Systems, vendar je bilo v začetku zanj zelo malo zanimanja, k čemur je pripomoglo tudi dejstvo, da je bila celotna dokumentacija na voljo le v kitajščini. Kasneje je podjetje AI-Thinker izdalo nov modul, imenovan ESP-01, ki je v računalniški stroki vzbudil nekoliko več zanimanja, s čimer je narasla tudi potreba po prevajanju dokumentacije v angleščino. Na trgu je več različic modula ESP8266 (tabela 2.1).

Trenutno je najbolj pribljubljen modul ESP-12e, ki smo ga uporabili pri našem delu (slika 2.1). Razlog za njegovo priljubljenost leži v večjem številu priključkov in boljšem dometu brezžičnega omrežja. Konec leta 2016 je bila

izdana tudi različica ESP-32, ki ponuja še večje število priključkov in povezavo Bluetooth ter procesor s frekvenco delovanja do 240 MHz.



Slika 2.1: ESP8266, različica 12e. Vir slike: [13]

Modul ESP-12e (v nadaljevanju modul ESP) ponuja kar 4 MB vgrajenega hitrega pomnilnika, poganja pa ga 32-bitni procesor s frekvenco 80 MHz, ki jo lahko povišamo na 160 MHz. Za povezovanje v brezžično omrežje Wi-Fi uporablja protokol IEE 802.11 b/g/n z različnimi varnostnimi protokoli WEP, WPA in WPA2. Komunikacija lahko poteka prek protokola TCP ali UDP. Modul lahko deluje v načinu dostopne točke in v načinu odjemalca omrežja Wi-Fi. Na voljo imamo 24 programsko nastavljivih priključkov. Modul podpira tudi različne protokole za komunikacijo z zunanjimi napravami (npr. SPI, I^2C, I^2S, UART) in 10-bitni analogno digitalni pretvornik.

Modul ESP programiramo s pomočjo serijskega pretvornika USB s priklopom na določene priključke na modulu. Na voljo imamo več odprtokodnih razvojnih okolij (npr. NodeMcu, microPython), najbolj priljubljeno pa je integrirano razvojno okolje Arduino, ki ima zelo veliko podporno skupino, programiranje pa poteka v jeziku C++.

Model	Število priključkov	Velikost hitrega pomnilnika
ESP-01, 02, 11	8	512 KB
ESP-03, 04	14	512 KB
ESP-05	5	512 KB
ESP-06	12	512 KB
ESP-07	16	1 MB
ESP-09	12	1 MB
ESP-12	16	4 MB
ESP-12E	22	4 MB
ESP-32	28	4MB

Tabela 2.1: Primerjava različic modula ESP8266

2.2 Vhodno-izhodne naprave

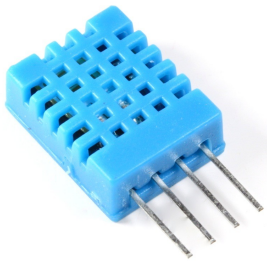
Vhodno-izhodne naprave so uporabljene za zajem podatkov iz okolice vozlišča in upravljanje z eno zunanjo napravo. Tipala omogočajo merjenje temperature, vlage, osvetlitve in zaznavanje gibanja. Za vklop ali izklop zunanje naprave je uporabljeno relejsko stikalo.

2.2.1 Tipalo za temperaturo in vlago DHT11

DHT11 (slika 2.2) je majhno in relativno poceni tipalo za merjenje temperature in vlage [4]. Deluje pri napetosti 3,3-5 V in na izhodu pošilja digitalni signal tako, da ni potrebe po analogno digitalnih pretvornikih. Za merjenje uporablja kapacitivno tipalo vlage in termistor[1] za merjenje temperature.

Tipalo deluje v temperaturnem območju 0-50 °C in v območju relativne vlažnosti 20-95 %, zato je primernejše za meritve v notranjih objektih, kjer temperatura ne pade pod 0 °C. Obstaja tudi boljša in natančnejša različica DHT22, ki ima večje območje meritve in natančnejši rezultat.

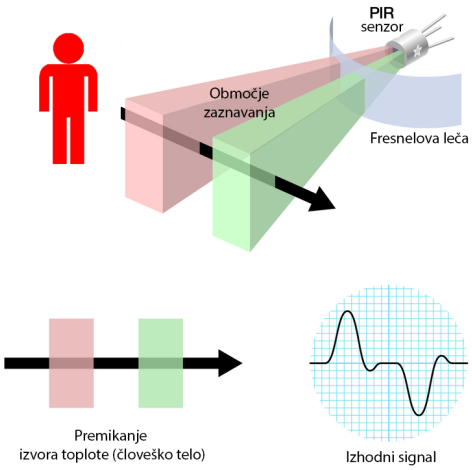
[1]termistor - element, ki spreminja svojo upornost na podlagi temperature



Slika 2.2: Tipalo DHT11. Vir slike: [15]

2.2.2 Infrardeče tipalo gibanja HC-SR501

HC-SR501 je tipalo gibanja, ki deluje na podlagi merjenja infrardečega sevanja [5]. V praksi to deluje tako, da je tipalo razdeljeno na dva dela, pri čemer prvi meri pozitivno spremembo infrardečega sevanja, drugi pa negativno spremembo, ki ga oddaja človeško telo. Ko se človek sprehodi mimo tipala, s tem najprej vpliva na prvo polovico tipala, kar sproži zaznavo pozitivne spremembe, nato pa na drugo polovico, kar sproži zaznavo negativne spremembe signala (slika 2.3). Te spremembe signalov se nato izmerijo.



Slika 2.3: Delovanje tipala gibanja. Vir slike: [5]

Če v prostoru ni premikajočega se človeškega telesa, obe polovici tipala zaznavata enako meritev infrardečega sevanja, zato ni spremembe v signalu. Tipalo gibanja deluje pri napetosti 5-20 V in na izhod vrača digitalni signal pri napetosti 0 ali 3,3 V. Območje zaznavanja je kot 120 ° pri razdalji 7 metrov. Tipalo ima tudi dva potenciometra[2], s katerima lahko nastavljamo občutljivost tipala in čas odprtja po zaznanem gibanju. Tipalo je pokrito z lečo, ki ima to lastnost, da razširi vidno polje zaznavanja infrardečega sevanja.

2.2.3 Fotoupor

Fotoupor je element, katerega upornost se spreminja glede na osvetljenost. Več kot je prejete svetlobe, manjšo upornost ima. Zato ga običajno povežemo tako, da merimo spremembo padca napetosti na fotouporu in s tem posredno tudi osvetljenost.

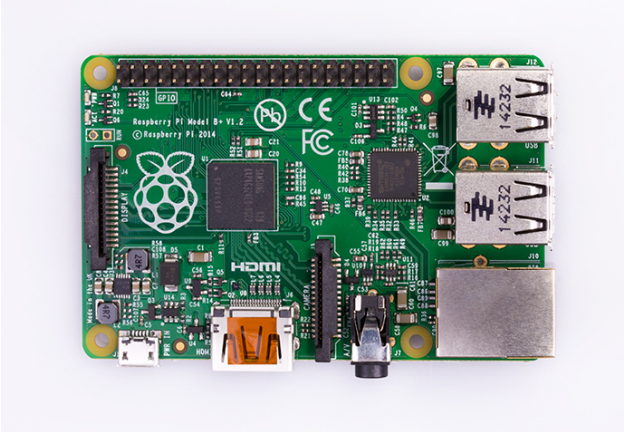
2.2.4 Relejsko stikalo Songle

Relejsko stikalo znamke Songle deluje pri napetosti 5 V. Rele je elektromagnetno stikalo, ki mehansko vklaplja ali izklaplja zunanjega porabnika s pomočjo tuljave. Če je v tuljavi napetost, je tokokrog na preklopni strani sklenjen, v nasprotnem primeru pa prekinjen. Rele Songle ima na preklopni strani možnost priklopa električne napetosti do 250 V in tokovne obremenitve do 10 A, kar zadošča za krmiljenje večine hišnih naprav.

2.3 Računalnik Raspberry PI 1 model B+

Za osrednji del sistema smo uporabili Raspberry PI 1 model B+ (slika 2.4). To je majhen računalnik velikosti bančne kartice, a dovolj zmogljiv za namestitev operacijskega sistema Linux verzije Raspbian z dodatkom za hišno avtomatizacijo Hassbian.

[2]potenciometer – element, ki mu lahko spreminjamo upornost



Slika 2.4: Raspberry Pi 1. generacije, model B+. Vir slike: [14]

Raspberry PI B+ prve generacije deluje na enojedernem 32-bitnem ARM procesorju s frekvenco 700 MHz [19]. Za procesor je uporabljen Broadcomov čip BCM2835. Pomnilnik je Samsungov velikosti 512 MB, ki se deli na delovni in grafični del. Operacijski sistem se naloži na kartico mikro SD (angl. Secure Digital card), ki ima lahko največ 64 GB prostora. Za priklop vhodno-izhodnih naprav imamo na voljo štiri priključke: USB 2.0, mrežni priključek, priključek HDMI (angl. High-Definition Multimedia Interface) in priključek za slušalke. V primeru uporabe računalnika Raspberry Pi za krmiljenje električnih naprav imamo na razpolago še 40 digitalnih programsko nastavljivih vhodov ali izhodov, ki delujejo pri maksimalni napetosti 3,3 V. Raspberry Pi napajamo s pomočjo priključka USB-mini z napetostjo 5 V.

Poglavje 3

Načrtovanje, izdelava in testiranje vezja vozlišča

Cilj predstavljenega dela je narediti majhno vezje na tiskani ploščici, ki bo vsebovalo vse potrebne vhodno-izhodne naprave, razen tipala gibanja in tipala osvetljenosti, ki sta priključena na vezje z dodatno žično povezavo. Tipalo gibanja in osvetljenosti sta tako lahko na zunanjosti ohišja. Prav tako mora biti napajanje prek priključka USB-mini. Potrebno pa je upoštevati naslednje lastnosti komponent:

- izhodi modula ESP delujejo pri napetosti 3,3 V,
- relejsko stikalo se odpre pri napetosti 5 V,
- fotoupor lahko na izhod vrača napetost med 0 in 1 V, saj ima modul ESP le en vhodno-izhodni priključek za branje analogne vrednosti, ki deluje le med zgoraj omenjenima napetostima.

Za realizacijo vezja smo uporabili program Altum Designer, namenjen načrtovanju in izdelavi električnih vezij. V njem si lahko izdelano vezje pogledamo v 3D obliki in vnaprej odpravimo napake pri sestavi vezja [1]. Omenjeni program vsebuje veliko bazo električnih komponent različnih proizvajalcev, ki jo po potrebi lahko nadgradimo. Vezje nam vrne v različnih oblikah zapisov, ki so primerne za izdelavo ploščice oziroma tiskanine. Izdelali

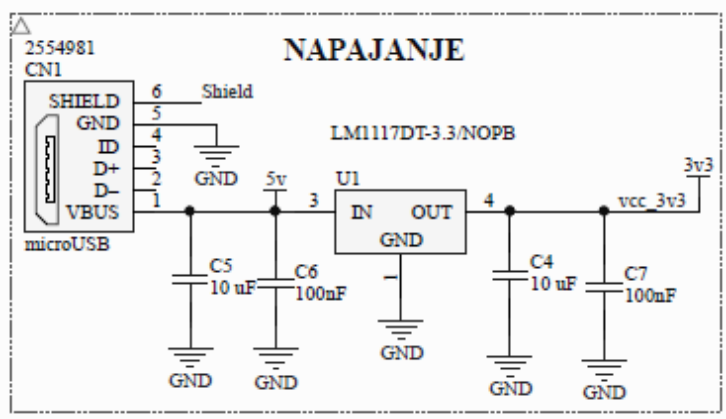
smo dve različici vezja, saj se je prva po testiranju izkazala za pomanjkljivo, z novo različico pa smo te napake odpravili.

3.1 Načrtovanje vezja

Pri načrtovanju vezja je potrebno dobro premisliti, kako posamezne komponente delujejo in kam jih postaviti na vezje. Dober načrt pomeni veliko manj popravkov pri naslednjih različicah vezja. V naslednjih podpoglavjih so opisani načrti posameznih delov vezja vozlišča.

3.1.1 Električno napajanje

Pri načrtovanju je bilo potrebno upoštevati napetostne razlike, kajti nekatera tipala in relejsko stikalo delujejo pri napetosti 5 V, modul ESP in ostala tipala pa pri napetosti 3,3 V. Na vezju je zato regulator napetosti LM1117DT, ki lahko na vhod sprejme napetost do 15 V in jo zmanjša na 3,3 V z maksimalnim tokom 800 mA, kar je več kot dovolj za modul ESP in tipala (slika 3.1).

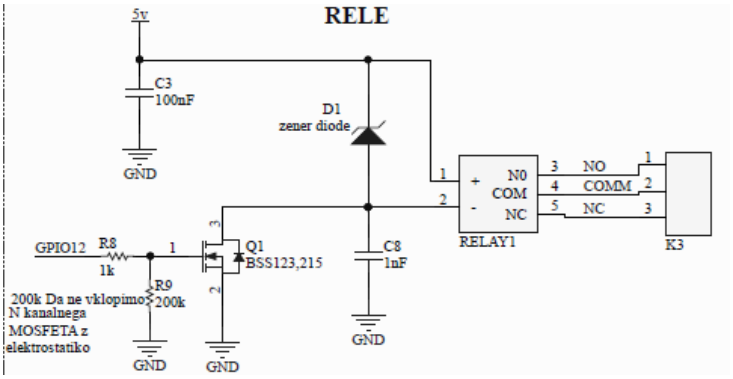


Slika 3.1: Shema napajalnega dela vezja

Pri vtičnici USB so uporabljeni samo priključki, ki so pod napetostjo. Po splošnih priporočilih so v nekatere dele vezja dodani še kondenzatorji, ki ublažijo nihanja napetosti. Napetost 5 V gre torej na tipalo gibanja in relejsko stikalo. Ostale komponente dobivajo 3 V iz napetostnega regulatorja.

3.1.2 Relejsko stikalo

Relejsko stikalo potrebuje za vklop tok 89,5 mA pri napetosti 5 V, kar je preveč za modul ESP, ki deluje pri maksimalni napetosti 3 V na GPIO priključkih. Za vklop releja skrbi tranzistor MOSFET [1], ki sklene negativni priključek z ozemljitvenim potencialom (glej sliko 3.2).



Slika 3.2: Shema relejskega stikala

Za vklop tranzistorja MOSFET skrbi priključek na modulu ESP. Zaradi varnosti je za zaščito pred napetostnimi sunki dodana dioda, ki preprečuje povratne sunke ob vklopu relejskega stikala, kar bi lahko poškodovalo modul ESP. Pred priključkom na tranzistor MOSFET je tudi 1 kΩ upor, ki skrbi za zaščito zaradi oscilacije, spodnji 200 kΩ upor pa skrbi za zaščito pred nezaželenim vklopom tranzistorja MOSFET s statično elektriko. Na stikalnem delu relejskega stikala je dodana še vrstna sponka za lažjo vezavo žic.

[1] Tranzistor MOSFET – tranzistor z učinkom električnega polja

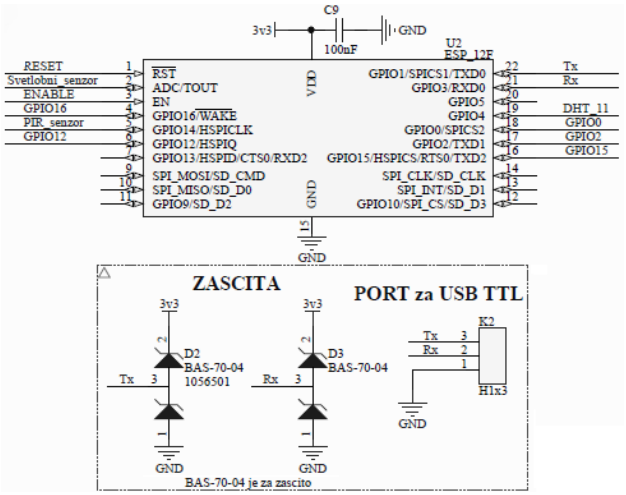
3.1.3 Tipalo osvetljenosti

Modul ESP ima le en analogni priključek, ki deluje v napetostnem območju med 0 in 1 V. Tukaj je potreben uporovni delilnik, narejen iz dveh zaporedno vezanih uporov. Po izračunu enačbe 3.1 potrebujemo za nižanje napetosti U2 z območja med 0 in 3,3 V na napetost U1 med 0 in 1 V dva upora, in sicer vrednosti R1=22 kΩ ter R2=10 kΩ.

$$U1 = (R2/(R1 + R2)) * U2 \tag{3.1}$$

3.1.4 Vezava modula ESP na vezju

Modul je nameščen na vezje s povezanimi priključki, ki so potrebni za normalni zagon (slika 3.3). Če želimo modul programirati, je dodano stikalo, ki ob pritisku sklene tokokrog med ozemljitvenim potencialom in priključkom GPIO0 ter se ob priključitvi na napetost postavi v način programiranja. Priključek reset je z 10 kΩ uporom povezan na priključek GPIO16, saj le-ta po preteku časovnika v načinu globokega spanja sproži signal za bujenje, kar sproži ponovni vklop.



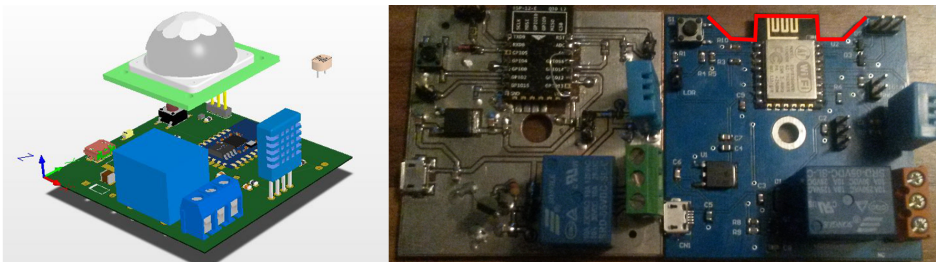
Slika 3.3: Vezava modula ESP in zaščite za serijsko komunikacijo

Tipalo gibanja in DHT11 sta vezana neposredno na priključke GPIO, in ker vračata digitalni signal z maksimalno napetostjo 3,3 V, ni potrebe po kakšnih dodatnih spremembah. Priključek Enable potrebuje za normalni zagon napetost 3,3 V.

Zaščita za serijsko komunikacijo je izdelana iz dveh elementov BAS70-04. Elementa zavarujeta modul ESP pred napačno priključitvijo serijskega komunikatorja na vezje vozlišča. Element BAS70-04 vsebuje dve diodi, ki so obrnjene vsaka v drugo smer. Element, ki je nameščen na priključek Tx, prepušča signal le v zunanjo smer (priključek Tx je oddajnik). Na priključku Rx pa je element obrnjen proti modulu ESP in prepušča le signal, ki pride na vezje vozlišča iz zunanjega komunikatorja (priključek Rx je sprejemnik).

3.2 Izdelava tiskanega vezja

Tiskano vezje je bilo strojno izdelano, komponente pa pritrjene kasneje. Izdelani sta bili dve različici; pri prvi so se pojavile težave z modulom ESP – nekateri priključki namreč niso bili pravilno načrtovani. V drugi različici so se te težave odpravile, pod anteno modula ESP pa se je po nasvetu kolega elektrotehnika dodalo izrez v vezju, kar je izboljšalo domet brezžičnega omrežja (izrez je označen z rdečo barvo, slika 3.4). Dodan je bil tudi odprti priključek, povezan neposredno na modul ESP, ki se ga lahko uporabi za vklop, izklop ter regulacijo led trakov.



Slika 3.4: Na levi 3D načrt, v sredini prva verzija in na desni druga ter hkrati zadnja verzija končanega vezja

3.3 Testiranje tiskanega vezja

Pri testiranju je potrebno izmeriti napetosti vseh elementov na vezju, pravilne napetosti na priključkih modula ESP in morebitne napake, ki bi lahko nastale pri spajkanju zaradi možnosti kratkega stika. V drugem delu je testirana električna poraba, zanesljivost in domet brezžičnega omrežja. Oprema je pri vseh testih enaka, brezžično omrežje pa je oddajal usmerjevalnik Linksys wrt54gl z naloženim sistemom DD-WRT [2]. Za osrednji strežnik je uporabljen Raspberry Pi (pogl. 2.3) z operacijskim sistemom Raspbian.

3.3.1 Poraba električne energije

Testirali smo električno porabo v različnih stanjih modula. Za napajanje je poskrbel polnilec za telefon z izhodom USB mini, ki deluje pri napetosti 5 V. Meritve so bile opravljene z multimetrom, vezanim zaporedno med priključkom USB in vtičnico USB.

- Način delovanja z vklopljeno povezavo Wi-Fi in sprejemanjem podatkov - poraba okrog 70 mA.
- Način delovanja z vklopljenim lahkim spanjem (angl. light sleep) - poraba med pošiljanjem naraste na 75 mA, v stanju lahkega spanja pa okrog 15 mA.
- Način globokega spanja (angl. deep sleep) - poraba se zniža na 0.12 mA za čas, ki ga uporabnik programsko določi.

Lahko spanje je način delovanja, v katerem se povezava Wi-Fi izklopi, procesor, sistemska ura in realno-časovna ura ostanejo aktivni. Ta način delovanja je že programsko vklopljen, če modul ESP nastavimo v način sprejemnika signala Wi-Fi.

[2]DD-WRT - odprtokodna strojno-programska oprema za brezžične usmerjevalnike in dostopne točke

Globoko spanje je način, v katerem se procesor, povezava Wi-Fi in sistemska ura izklopijo, aktivna ostane le ura realnega časa, ki nato modul nazaj zbudi. Pri tem načinu je potrebno priključek, ki pošlje signal bujenja, povezati s priključkom reset, med njima moramo še zaporedno vezati upor, saj se v nasprotnem primeru modul ne zažene znova. V načinu globokega spanja je poraba najmanjša in je zato zelo uporaben za samostojne sisteme, pri katerih se modul ESP napaja z baterije.

Za preizkus napajanja z baterije smo uporabili prenosno polnilno baterijo USB za polnjenje telefonov. Napetost baterije je 5 V s kapaciteto 2200 mAH. Po izračunih bi modul ESP ob porabi brez načina globokega spanja zdržal približno 31 ur po naslednjem izrazu

$$T = 2200mAH/70mA = 31.42ur. \tag{3.2}$$

Pri uporabi načina globokega spanja bi se čas bistveno podaljšal po naslednjem izrazu

$$T = 2200mAH/0.15mA = 14666ur = 611dni = 1.674let. \tag{3.3}$$

Praktični preizkus z baterijo je potekal ob pošiljanju podatkov na osrednji strežnik z merjenjem časa. Podatki so se pošiljali na vsako minuto. Zadnji podatek je bil na strežnik poslan po 26 urah in 31 minutah.

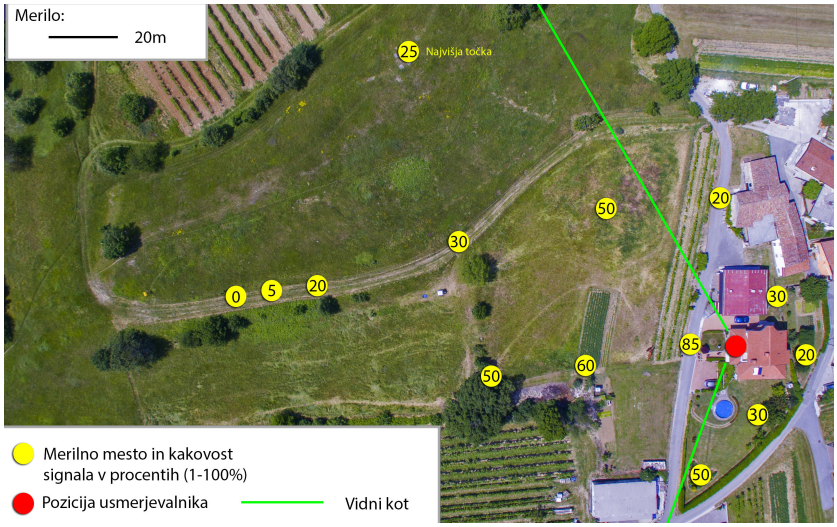
3.3.2 Domet brezžičnega omrežja

Meritve dometa so potekale s prenosno enoto napajanja za vozlišče in izpisom kakovosti signala s pomočjo serijskega pretvornika na računalnik. Za oddajanje omrežja je skrbel usmerjevalnik Linksys wrt54gl, modul ESP pa je vsakih 5 sekund v odstotkih od 0 do 100 sporočal kakovost signala po izrazu

$$Q = 2 * (dBm + 100), \tag{3.4}$$

ki ga uporablja podjetje Microsoft za izračun kakovosti brezžičnega signala. Izraz za izračun kakovosti signala potrebuje spremenljivko dBm, ki nam jo modul ESP vrača v programski kodi kot moč signala v decibel milivatih.

S prenosno enoto smo merili signal na različnih lokacijah okrog dostopne točke Wi-Fi. Signal je bil najboljši v vidnem kotu usmerjevalnika, presenetila pa nas je razdalja od usmerjevalnika, kjer je bil signal še vedno sprejet – ta je bila namreč večja, kot smo pričakovali (slika 3.5).



Slika 3.5: Zemljevid merilnih mest z izpisom kakovosti signala

3.3.3 Stabilnost povezave Wi-Fi

V objektu sta bili nameščeni dve vozlišči, in sicer vsako v svojem nadstropju z usmerjevalnikom v sredinski etaži. Prvi korak je bilo testiranje prekinitev brezžičnega omrežja oziroma izguba signala s strani modula. Skripta je preverjala stanje povezave in ob vsaki prekinitvi to zabeležila.

Testiranje je potekalo 3 dni in v tem času se je na obeh modulih ESP povezava z usmerjevalnikom prekinila samo enkrat. Preverili smo, ali je bil za prekinitev morda kriv izpad usmerjevalnika, vendar do prekinitve pri modulih ni prišlo istočasno, iz česar je logično sklepati, da je bil krivec modul ESP. Testirali smo tudi načrtni izklop usmerjevalnika, a se je izkazalo, da sta se modula brez težav povezala nazaj takoj, ko je bilo to mogoče.

Modul ESP podpira dva omrežna kontrolna protokola za pošiljanje po

omrežju, in sicer TCP in UDP. Za testiranje le-teh smo v shemo testiranja vključili še Raspberry Pi, ki je poganjal skripto v jeziku Python. Vse skupaj je nato povezal brezžični usmerjevalnik Linksys wrt54gl.

Za namen testiranja zanesljivosti protokolov se je na modulih ESP za vsak protokol več dni izvajala skripta, ki je na določenih vratih pošiljala pakete na Raspberry Pi, ta pa je pakete prebral in jih shranjeval v tekstovne datoteke. Paketi so se pošiljali v obliki zaporedne številke paketa, ure, datuma in omrežnega naslova, s katerega je bil paket poslan. Pošiljanje je potekalo na vsakih trideset sekund. Pri vsakem paketu je bilo dodano še število izgub povezave Wi-Fi, saj bi v nasprotnem primeru lahko krivdo pripisali protokolu, modul ESP pa bi bil zaradi prekinitve delovanja usmerjevalnika v tistem trenutku brez povezave. Postavitve vozlišč so bile enake kot pri testiranju povezave Wi-Fi, kot opisano zgoraj.

Protokol UDP se je odrezal slabše kot TCP. Razlog za to je v tem, da UDP ne kontrolira prejetih paketov. Pošiljatelj in sprejemnik namreč ne vzpostavita povezave in pošiljatelj ne preverja, ali je sprejemnik pakete prejel. Pri 6827 paketih se jih je izgubilo 31, kar predstavlja 0,45 % celote. Dobra stran protokola UDP pa je večja hitrost pošiljanja. Protokol TCP se je odrezal mnogo bolje kot UDP pri zanesljivosti prenosa. Do izraza je prišla kontrola poslanih paketov – pri 12135 poslanih paketih ni prišlo do izgub. Hitrost pošiljanja je bila sicer manjša, zato pa je bil prenos zanesljivejši. TCP smo testirali 4 dni, UDP pa 3 dni.

Poglavje 4

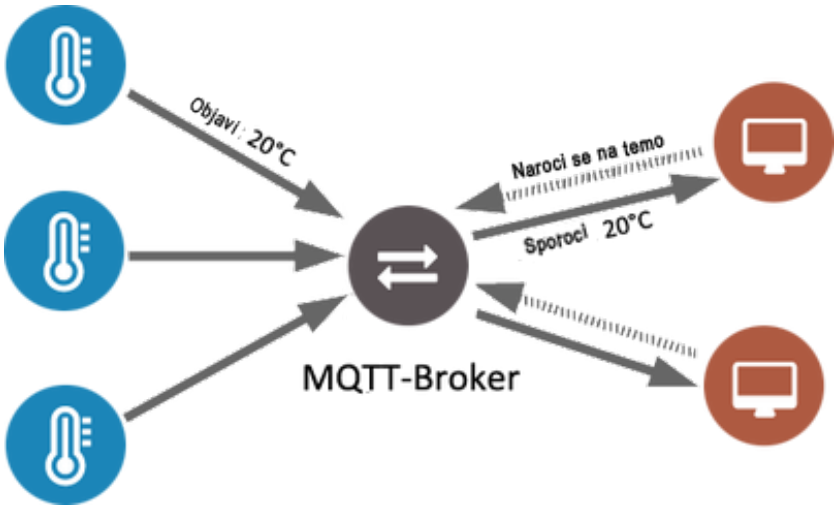
Vzpostavitev osrednjega avtomatizacijskega sistema

V pametni hiši morajo vse naprave delovati usklajeno in vsaka sprememba na napravah mora biti vidna. V današnjih časih, ko imamo skoraj vsi pametne telefone, je nujno, da lahko hišno avtomatizacijo spreminjamo iz vsake pametne naprave in je sprememba vidna v celotnem sistemu na vseh napravah v omrežju. Tukaj pa ne smemo pozabiti še na željo po hitrosti delovanja in varnosti sistema pred vdori. Nekateri se odločijo za zaprte sisteme, ki so vidni samo znotraj domačega omrežja in so brez možnosti internetnega dostopa. Drugi želijo imeti tudi oddaljen dostop, za kar je nujen dostop do interneta. Z vsemi zgoraj omenjenimi parametri v mislih smo poiskali protokol, ki bi bil primeren za vozlišča in bi deloval po brezžičnem omrežju Wi-Fi. Za komunikacijo vozlišča z osrednjim sistemom smo izbrali protokol Mqtt. Za osrednji sistem je bilo potrebno poiskati programsko opremo, ki lahko deluje na računalniku Raspberry Pi in omogoča uporabo protokola Mqtt. Sistem mora biti dosegljiv na pametnih telefonih in računalnikih. Kot ustrezen se je izkazal osrednji sistem za hišno avtomatizacijo Home Assistant.

4.1 Komunikacijski protokol Mqtt

Mqtt (angl. Message Queuing Telemetry Transport) je odprt, preprost in hiter protokol za komunikacijo med napravami s pomočjo posrednika (angl. broker) [21]. Razvila sta ga Andy Stanford-Clark in Arlen Nipper leta 1999 [10]. Cilj je bil razvoj protokola, ki ima učinkovito uporabo pasovne širine in porabi malo električne energije, saj so bile naprave povezane prek satelitskih povezav in je bil prenos podatkov v tistem času zelo drag.

Za delovanje potrebuje samo lokalno omrežje ter strežnik, na katerem deluje posrednik Mqtt. Odjemalci se povežejo na osrednji strežnik in poslušajo ali oddajajo sporočila, ki so povezana z določeno temo (angl. topic, slika 4.1). Strežnik poskrbi, da se vsako sporočilo, povezano z določeno temo, pošlje vsem odjemalcem, ki to temo poslušajo. Vsaka sprememba teme je vidna vsem naročenim odjemalcem. Sporočila so zelo kratka in hitra ter delujejo po protokolu TCP na vratih 1883. Za vsako poslano sporočilo pošiljatelj prejme povratnico o pravilni in uspešni dostavi.



Slika 4.1: Delovanje protokola Mqtt. Vir slike: [12]

Posrednik hrani vse v tistem trenutku povezane naprave in jim vsake toliko časa pošlje sporočilo, na katerega se morajo odzvati, v nasprotnem

primeru jih izbriše s seznama povezanih naprav. Odjemalec se lahko prijavi na več tem, s katerih sprejema podatke. Pošiljanje na več tem hkrati je prav tako možno. Teme so sestavljene nivojsko z vmesnimi poševnicami, npr. *hiša/soba1/stikalo*. Ob poslanem sporočilu na višji nivo, to sporočilo prejmejo tudi odjemalci, prijavljeni na nižjih nivojih. Nivoji si sledijo z leve proti desni, z najvišjim nivojem na levi ter vmesno poševnico.

Varnost protokola Mqtt je eno pogostejših vprašanj, ki se pojavi pri uporabi [18]. Ob namestitvi je protokol Mqtt odprt ter brez varnostnih mehanizmov, ki bi preprečevali nekomu v lokalnem omrežju komunikacijo s strežnikom Mqtt, objavo sporočil na teme ali pa branja le-teh. Obstaja več varnostnih mehanizmov in dobro jih je vsaj nekaj od naštetih uporabiti:

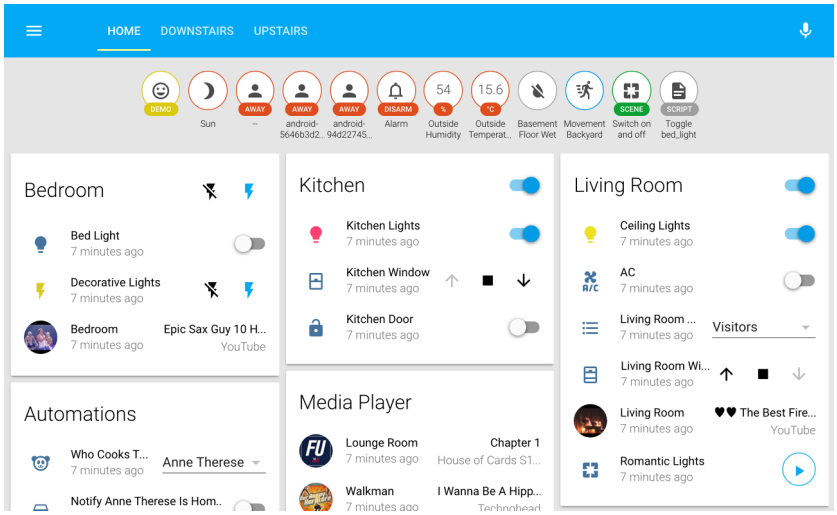
- uporabniško ime in geslo,
- enkripcija sporočil SSL/TLS,
- uporaba navideznega zasebnega omrežja (VPN) za oddaljen dostop,
- uporaba certifikata za komunikacijo.

Priporočeno je, da se pri vsakem sistemu z Mqtt uporabi vsaj uporabniško ime in geslo. Če smo povezani v omrežje, sporočil Mqtt ni težko prestreči, zato je dobro namestiti tudi enkripcijo SSL/TLS, ki sporočila Mqtt zakriptira na pošiljateljevi strani ter odkriptira na prejemnikovi strani. Za še boljšo varnost pa je vsekakor potrebna uporaba certifikata za komunikacijo. Težava nastane, ker večina krmilnikov ni dovolj zmogljiva in hitra za kriptiranje sporočil Mqtt ali za uporabo certifikata pri komunikaciji. Če želimo imeti oddaljen dostop do sistema hišne avtomatizacije, je priporočljivo, da uporabljamo navidezno zasebno omrežje VPN (angl. Virtual private network).

4.2 Osrednje programje Home Assistant

Home Asisstant je odprtokodni program za hišno avtomatizacijo [7]. Omogoča pregled in kontrolo nad pametnimi napravami in tipali v hiši. Program je do-

volj kompakten, da lahko deluje na računalniku Raspberry Pi in hkrati dovolj zmogljiv za naprednejšo avtomatizacijo, kot so urniki prižiganja, analiza vremena, sledenje določenim napravam v bližini hiše, avtomatska kontrola nad ogrevanjem, kamere ter mnogo drugih možnosti delovanja (slika 4.2). Prednost programa je tudi možnost uporabe lastnih skript za avtomatizacijo. Če želimo glasovni nadzor nad svojo hišo, lahko Home Assistanta povežemo tudi s sistemi za glasovno kontrolo, kot so Amazon Echo in Google Assistant.



Slika 4.2: Uporabniški vmesnik programa Home Assistant. Vir slike: [16]

Za komunikacijo s tipali se lahko uporablja večina znanih protokolov za hišno avtomatizacijo, npr. Mqtt, Z-Wave, ZigBee. Ker je program odprtokodni, ga lahko uporabnik poljubno razvija in prilagaja po lastnih željah, zato obstaja na spletu ogromna baza posodobljenih primerov ter novih protokolov. Na voljo je tudi mobilna aplikacija za pametne telefone, druga možnost pa je dostop s spletnim brskalnikom, saj je uporabniški vmesnik načrtovan za mobilne naprave in računalnike z večjimi zasloni.

4.3 Namestitev in priprava sistema

Namestitev sistema Home Assistant oziroma Hassbiana (ker je zasnovan na sistemu Raspbian, so ga poimenovali Hassbian) poteka tako, da sliko sistema naložimo na mikro pomnilniško kartico in jo vstavimo v računalnik Raspberry PI. Za pravilno namestitev je potrebna povezava z internetom. Po zagonu sistem potrebuje približno 20 minut, da se pripravi za delovanje. Po tem času lahko do uporabniškega vmesnika dostopamo z druge naprave preko naslova IP in vrat 8123.

Po postavitvi Home Assistanta smo namestili še strežnik Mqtt, imenovan Mosquitto. Mosquitto je odprtokodni brezplačni strežnik Mqtt za uporabo na operacijskih sistemih Linux [9]. Namestimo ga z ukazom *apt-get install mosquitto* v ukazno vrstico (angl. terminal). Po namestitvi je potrebno zagnati še ukaz *apt-get update*, ki nam posodobi vse potrebne knjižnice za strežnik Mqtt. Konfiguracija Home Assistanta se nahaja v domačem direktoriju, zato je priporočljivo, da na Raspberry PI namestimo še strežnik Samba, ki omogoča lokalni dostop do datotek na sistemu, saj Raspberry PI nima zaslona.

V konfiguracijsko datoteko vpisujemo vse naprave, ki bodo uporabljene v hišni avtomatizaciji. Ker bo za komunikacijo med Home Assistantom in vozliščem uporabljen protokol Mqtt, je potrebno v datoteko dodati konfiguracijo Mqtt, ki jo lahko dobimo na spletni strani Home Assistanta.

```
mqtt:
  broker: localhost
  port: 1883
  client_id: home-assistant
  username: homeassistant
  password: homeassistant
```

Ker strežnik Mqtt in osrednji sistem Home Assistant delujeta na istemu računalniku, je za naslov strežnika IP potrebno uporabiti oznako "localhost". Uporabniško ime in geslo lahko na Raspberry PI s pomočjo ukazne vrstice

določimo sami. Home Assistant ima tudi možnost svojega strežnika Mqtt, vendar smo raje uporabili strežnik Mosquitto, ker je le-ta novejši in ga lahko upravljamo z ukazne vrstice, hkrati pa deluje neodvisno od sistema Home Assistant.

Vsako napravo, ki jo želimo upravljati ali brati z nje podatke, je potrebno v novi vrstici dodati v konfiguracijsko datoteko. Na spletni strani imamo primere za raznorazne naprave, ki jih lahko Home Assistant upravlja, npr. stikala, luči, temperaturna tipala ipd. Po spremenjeni konfiguraciji je potrebno sistem ponovno zagnati oziroma vsaj v uporabniškem vmesniku ponovno zagnati program Home Assistant.

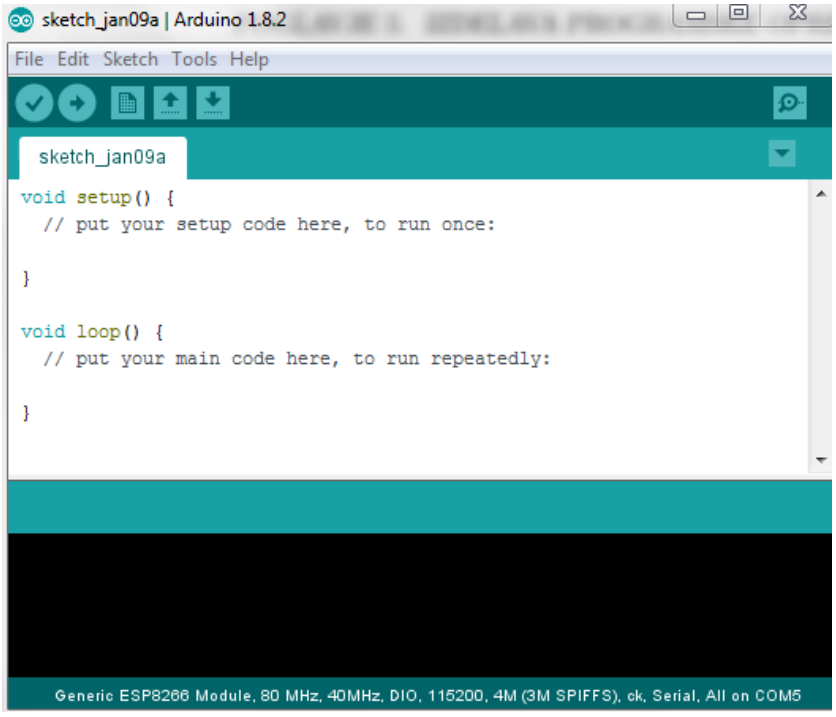
Poglavje 5

Izdelava programske opreme vozlišča

Izdelava programske opreme je zahtevala premišljen pristop k načrtovanju, saj je bil eden od ciljev izdelek, ki je preprost za uporabo. Pri razvoju je bilo potrebno vključiti pogled skozi oči uporabnika, saj je to edini način, kako narediti izdelek, ki je za končnega uporabnika enostaven. Upoštevati je bilo potrebno, da končni uporabnik nima programerskega znanja, ne ve, kako vozlišče deluje, vendar ga kljub temu zna povezati v svoje brezžično omrežje in pripraviti za delovanje. Večina procesov je tako avtomatiziranih znotraj modula ESP, uporabniški vmesnik je izdelan na preprost, a uporabniku prijazen način, in dostopen prek spletnega brskalnika. Pri načrtovanju uporabniškega vmesnika je bilo potrebno upoštevati različne dimenzije zaslonov, s katerih bo uporabnik dostopal do vozlišča. Pri izdelavi vezja so bile že vse poti in tipala vnaprej načrtovani, zato spremembe le-teh niso bile mogoče. Zaradi uporabe protokola Mqtt za komunikacijo pa je bilo potrebno razmisliti, kako dodati možnost spreminjanja tem, na katere vozlišče posluša ali pa na njih pošilja podatke. V programski kodi so uporabljene nekatere javno dostopne knjižnice (npr. Json, AsyncWebServer, SPIFFS), ostalo pa je v celoti izdelano v okviru diplomske naloge.

5.1 Priprava razvojnega okolja

Programiranje je potekalo v razvojnem okolju Arduino IDE (slika 5.1) ter v programskem jeziku C++ [2]. Pred programiranjem modula ESP je potrebno s spleta dodati aktualno strojno-programsko opremo modula ESP, ki jo razvijajo spletni uporabniki in jo lahko programiramo v razvojnem okolju Arduino v programskem jeziku C++ .



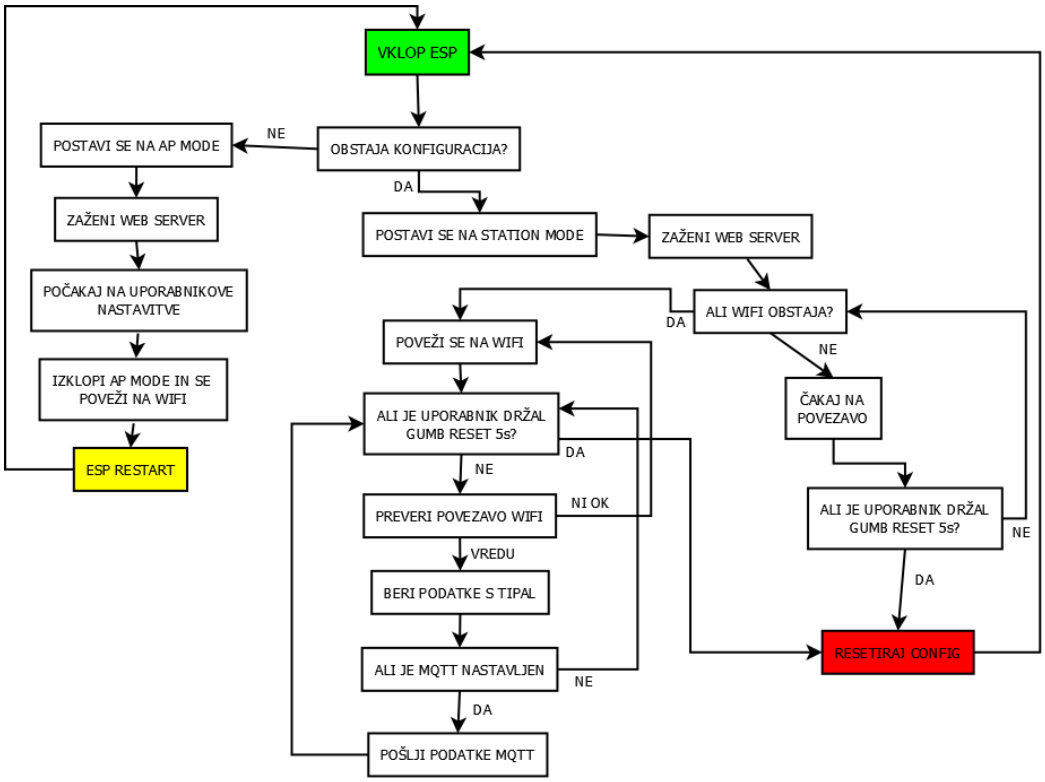
Slika 5.1: Integrirano razvojno okolje Arduino IDE

Integrirano razvojno okolje Arduino IDE vsebuje veliko zbirko knjižnic, ki jih lahko dodajamo po potrebi. Za pregled kode vsebuje prevajalnik, ki nam vrne napake v skripti. Slabost Arduino IDE-ja pa je v tem, da nima avtomatskih popravkov skripte in ponujenih predlogov pri pisanju skripte. Programiranje vozlišča poteka s pomočjo serijskega pretvornika, ki ga priključimo v vtičnico USB na računalniku in na priključke za programiranje na vozlišču. Ob priklopu napajalnika USB v vozlišče je potrebno držati gumb

za vklop programerskega načina (glej podpogl. 3.1.4).

5.2 Postopek ob vklopu vozlišča

Načrtovanje programske opreme smo začeli z diagramom poteka, kjer smo nanizali vsa stanja vozlišča od vklopa naprej in korake, ki jih vozlišče izvede (slika 5.2). Pri načrtovanju diagrama je potrebno paziti, da se nam kje ne pojavi neskončna zanka.



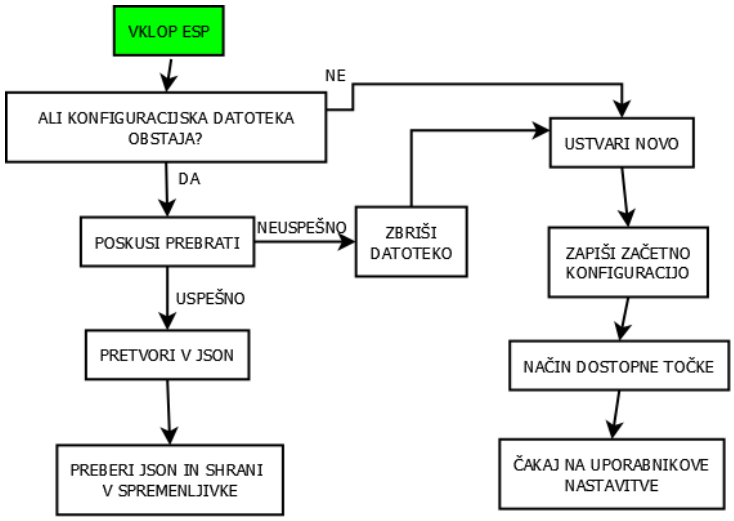
Slika 5.2: Diagram poteka ob vklopu napajanja vozlišča

Vozlišče je zasnovano tako, da ob vklopu v pomnilniku (angl. flash) preveri, ali obstaja konfiguracija omrežja in tipal. Če konfiguracije ni, se vozlišče postavi v način dostopne točke, na katero se uporabnik poveže s pametno napravo. Na prednastavljenem naslovu IP v spletnem brskalniku uporabnik

dostopa do vmesnika, kjer lahko nastavi novo omrežje in vklopi sprejemanje in pošiljanje podatkov prek Mqtt na strežnik. Po uspešno shranjenih nastavitvah se vozlišče poveže na novo omrežje in začne pošiljati podatke na nastavljen strežnik Mqtt. V primeru, da uporabnik želi vozlišče znova nastaviti, lahko konfiguracijo ponastavi s 5 sekund dolgim pritiskom na gumb na vezju, po katerem vozlišče izbriše svojo konfiguracijo in jo vrne na začetno.

5.3 Branje konfiguracije iz pomnilnika

Konfiguracija vozlišča se shranjuje v pomnilnik modula ESP v datoteki */config/config.json* v obliki Json. Za branje konfiguracije skrbi knjižnica SPIFFS, ki nam podatkovni sistem predstavi v mapah in datotekah kot pri običajnih operacijskih sistemih [17]. Modul ESP ob zagonu najprej prične branje iz pomnilnika s pomočjo knjižnice SPIFFS in preveri, ali konfiguracijska datoteka obstaja (glej sliko 5.3). Če datoteka ne obstaja, jo ustvari, v njo zapiše privzeto konfiguracijo in modul zažene v načinu dostopne točke.



Slika 5.3: Potek branja konfiguracije iz pomnilnika

V primeru, da datoteka obstaja, jo poskusi prebrati in preveri, ali je konfiguracija zapisana v pravilni obliki. V primeru uspešnega branja se prebrana

konfiguracija s pomočjo knjižnice Json pretvori v obliko (slika 5.4), ki jo lahko preberemo v lokalne spremenljivke [8]. Pri pretvorbi konfiguracije v obliko Json je prisoten tudi varnostni mehanizem, ki poskrbi, da je Json uspešno pretvorjen.

```
struct Configuration {
  String ssid,pass;
  bool dhcp;
  IPAddress ip,netmask,gateway;
  //konfiguracija naprave
  String device_name;
  bool deep_sleep;
  //interval branja podatkov 1000=1s
  long readinterval;
  //mqtt konfiguracija
  bool mqtt;
  IPAddress mqttserver;
  int mqttport=1883;
  String mqttusername,mqttpass;
  String mqtttopic,mqtttopictemp,mqtttopichum,mqtttopicrelay,mqtttopicmoving,mqtttopiclight;
};
```

Slika 5.4: Struktura v programu po prebrani konfiguraciji

Po pretvorbi se konfiguracija zapiše v lokalne spremenljivke, ki jih nato modul ESP uporablja za delovanje. Če v konfiguraciji ni shranjenih podatkov za povezavo Wi-Fi, se modul ESP avtomatsko postavi v način dostopne točke. V primeru praznega imena naprave, ki ga lahko uporabnik določi sam in s katerim se modul ESP predstavlja na omrežju, se v ime naprave zapišejo "WMS"in zadnje štiri številke naslova MAC. Celotno branje konfiguracije je narejeno v obliki podprograma, saj se v programu večkrat pokliče, rezultat podprograma pa je zastavica, ki glavni program oziroma klicatelja podprograma obvesti, ali je bila konfiguracija uspešno prebrana (glej sliko 5.5).

Po prebrani konfiguraciji se modulu ESP nastavijo vhodni in izhodni priključki, ki so povezani na naprave na vezju, nato se vozlišče postavi v način sprejemnika Wi-Fi ali dostopne točke, odvisno od nastavitev. V primeru dostopne točke se pokaže na listi oddajnikov Wi-Fi z imenom, ki je uporabniško ali pa avtomatsko določeno. Če sta ime omrežja in geslo nastavljena, se vozlišče poveže na določeno omrežje in skrbi za ponovno vzpostavitev po pre-

kinitvi povezave Wi-Fi. V primeru prekinitve se vozlišče znova poveže nazaj na omrežje, vsi ostali podprogrami so v vmesnem času na čakanju, razen podprograma za ponastavitev konfiguracije.

```
boolean loadConfig(){
  File c = SPIFFS.open(config_file, "r");
  if ( c.size() == 0 ) {
    Serial.println("Konfiguracijska datoteka je prazna in neveljavna!");
    SPIFFS.remove(config_file);
    checkConfig();
  }else{
    std::unique_ptr<char[]> buf (new char[size]);
    DynamicJsonBuffer jsonBuffer;
    JsonObject& configuration = jsonBuffer.parseObject(buf.get());
    if (!configuration.success()){
      LOG.println("Ne morem prebrati json datoteke");
      return false;
    }else{
      config.ssid=configuration["network"]["ssid"].as<String>();
      config.pass=configuration["network"]["pass"].as<String>();
      config.dhcp=configuration["network"]["dhcp"];
    }}}
```

Slika 5.5: Povzetek podprograma za branje konfiguracije

5.3.1 Ponastavitev konfiguracije vozlišča

Ponastavitev vozlišča je narejena tako, da ne prekinja delovanja ostalih podprogramov. Ob pritisku gumba za ponastavitev se sproži časovnik in program konstantno preverja, ali je gumb še vedno pritisnjen (glej sliko 5.6). Če se gumb med pritiskom spusti, se časovnik ponastavi na vrednost 0. Če vrednost časovnika med držanjem gumba naraste nad 5 sekund, se sproži varnostna zastavica, ki čaka na spust gumba, po katerem se konfiguracija ponastavi na privzete vrednosti in vozlišče se znova zažene. Varnostna zastavica skrbi za pravilni zagon vozlišča po ponastavitvi, saj bi se drugače modul ESP zagnal v načinu programiranja (podpogl. 3.1.4).

```
//glej ali je bil gumb reset pritisnjen
    if(reset_button_val == LOW && reset_button_state== HIGH){
      reset_dntime=currentMillis;
    }

    //ali je bil pritisnjen za vec kot reset_interval
    if(reset_button_val == LOW  && (currentMillis - reset_dntime) > reset_interval){
        Serial.println("Reset gumb pritisnjen za 5 sekund");
        reset_dntime = currentMillis;
        reset_button_state=false;
        reset_config_flag=true;
    }
      reset_button_state =  reset_button_val;

    if(reset_button_val!=LOW && reset_config_flag){
      Serial.println("Resetiram config");
      SPIFFS.remove(config_file);
      delay(1000);
      ESP.restart();
    }
```

Slika 5.6: Del podprograma za ponastavitev konfiguracije

5.3.2 Shranjevanje nove konfiguracije

Za shranjevanje nove konfiguracije skrbi podprogram, ki iz trenutnih nastavitev pridobi vse podatke iz konfiguracije 5.4. S podatkov nato z uporabo knjižnice sestavi objekt Json v dveh delih, od katerih je prvi konfiguracija omrežja Wi-Fi, drugi pa konfiguracija naprave. Tudi če vsi podatki v konfiguraciji niso izpolnjeni, se le-ti shranijo v objekt Json. Objekt Json se nato spremeni v obliko zaporedja, katero se nato prepiše v konfiguracijsko datoteko v pomnilniku (glej sliko 5.7). Prednost tega načina shranjevanja je v tem, da ni potrebno poklicati podprograma za branje shranjene konfiguracije z datoteke, saj je nova konfiguracija že aktivna in shranjena v lokalnih spremenljivkah. Po zapisu konfiguracije se objekt Json izbriše, saj tako prihranimo prostor v delovnem pomnilniku.

```
void saveFullConfig(){
    DynamicJsonBuffer jsonBuffer;
    JsonObject& root = jsonBuffer.createObject();
    JsonObject& network = root.createNestedObject("network");
    File c = SPIFFS.open(config_file, "w");
      if (!c) {
        Serial.println("Odpiranje in zapis v konfiguracijsko datoteko neuspesen");
        return;
      }
    network["ssid"] = config.ssid;
    network["pass"] = config.pass;
    network["dhcp"] = config.dhcp;
    String new_config;
    root.printTo(new_config);
    c.println(new_config);
    c.close();
    jsonBuffer.clear();
}
```

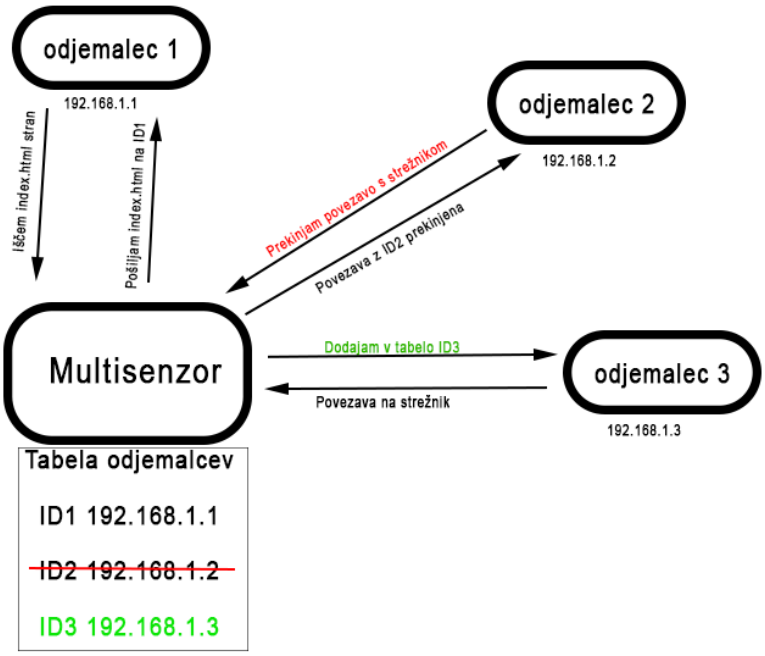
Slika 5.7: Povzetek podprograma za shranjevanje konfiguracije

5.4 Uporabniški vmesnik

Uporabniški vmesnik, shranjen v pomnilniku, deluje na lokalnem spletnem strežniku na modulu ESP. Za serviranje uporabniškega vmesnika smo izbrali javno dostopno knjižnico asinhronskega strežnika AsyncWebServer, ki hkrati lahko upravlja več odjemalcev uporabniškega vmesnika [6]. Strežnik deluje tako, da vsakega novega odjemalca doda v tabelo in zahteve obdeluje po vrsti od prve naprej. Vsak odjemalec je predstavljen z novo identifikacijsko številko, ki mu je dodeljena ob povezavi na strežnik vozlišča (slika 5.8).

Asinhronski strežnik ima možnost pošiljanja sporočil tako ob vzpostavljeni povezavi z odjemalcem kot tudi ob prekinjeni. Ko odjemalec pošlje sporočilo na strežnik, je to sporočilo predstavljeno z identifikacijsko številko odjemalca. Ob odjemalčevi zahtevi po spletni strani strežnik samodejno prebere pomnilnik modula ESP in vrne odgovor na zahtevek s spletno stranjo. Tako se na vozlišču v pomnilniku modula ESP nahaja celoten grafični vmesnik, narejen v obliki HTML z zaledjem Javascript. Ker se uporabniški vmesnik naloži pri odjemalcu, je za komunikacijo med modulom ESP in odje-

malčevo napravo potrebno odpreti spletni vtičnik, ki lahko z modulom ESP komunicira prek protokola TCP. Spletni vtičnik podpira programski jezik Javascript, ki je uporabljen za zaledje uporabniškega vmesnika.



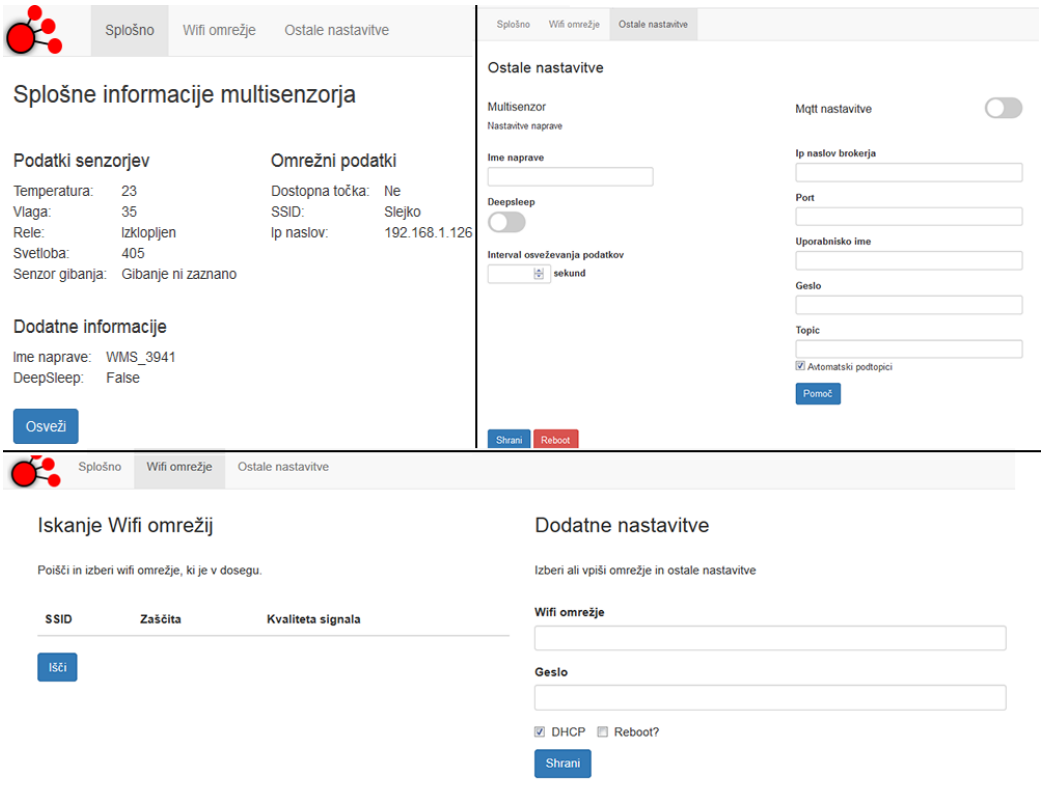
Slika 5.8: Delovanje asinhronskega strežnika

5.4.1 Izdelava vizualne podobe vmesnika

Zaradi želje po dostopu do grafičnega vmesnika v spletnem brskalniku, je bilo potrebno grafični vmesnik dojemati kot spletno aplikacijo. Zaradi nujnosti po prileganju vsem velikostim zaslonov smo za podlago uporabili Bootstrap.

Bootstrap je odprtokodno orodje za izdelavo dinamičnih spletnih strani, ki se spreminjajo glede na velikost zaslona [3]. Pri izdelavi je bilo potrebno paziti tudi na velikost datotek, saj je na modulu ESP velikost pomnilnika omejena. Omejili smo se na tri podstrani, v katerih so splošne informacije o vozlišču, iskanje in nastavitve omrežja Wi-Fi ter dodatne nastavitve vozlišča za komunikacijo (slika 5.9). Pri vnosih podatkov v vsa polja so narejeni

mehanizmi, ki preverjajo pravilnost vnosa.



Slika 5.9: Vizualni izgled uporabniškega vmesnika

V zaledju uporabniškega vmesnika je skripta Javascript skupaj s knjižnico Jquery, ki skrbi za vse uporabniške klike in vnose. Pri nastavitvah brezžičnega omrežja ima uporabnik možnost poiskati razpoložljiva omrežja, nastaviti svoj naslov IP vozlišča, pri dodatnih nastavitvah pa je možnost vklopa ali izklopa protokola Mqtt in dodajanja svojih tem, na katere naj vozlišče posluša in pošilja podatke. Uporabnik lahko spremeni tudi ime vozlišča, s katerim se predstavi v omrežju. V splošnih informacijah so podatki s tipal, ki se osvežijo ob kliku na gumb. Možna je tudi sprememba intervala pošiljanja podatkov ter možnost vklopa načina globokega spanja (podpogl. 3.3.1). Uporabniški vmesnik je zgoščen v formatu gz, ki je poznan novejšim brskalnikom, saj tako prihranimo veliko prostora v pomnilniku modula ESP in hkrati skrajšamo čas

nalaganja vmesnika na uporabnikovem brskalniku. Velikost datoteke vmesnika je 255 KB, po zgostitvi pa 86 KB.

5.4.2 Komunikacija med uporabniškim vmesnikom in vozliščem

Komunikacija poteka z uporabo spletnega vtičnika (angl. websocket) tako na strani vozlišča kot tudi uporabnika. Spletni vtičnik zagotavlja dvosmerno komunikacijo po protokolu TCP. Za komunikacijo se na obeh straneh uporablja ukaze v obliki črke in številke, v nekaterih primerih pa so na koncu dodani še podatki v obliki Json (tabela 5.1). Vsak ukaz se vedno začne s črko in številko. Obe strani imata narejen sprejemnik, ki ob prejetem sporočilu preveri, kakšen je ukaz, in nato podatke pošlje na podprogram, ki skrbi za obdelavo tega ukaza. Ob odprtju uporabniškega vmesnika se na modul ESP pošlje ukaz S1, ESP nato prebere podatke s tipal, podatke povezave Wi-Fi in splošne podatke o vozlišču, ter pošlje nazaj ukaz S1 skupaj z dodanimi podatki v obliki Json ('S1' + podatki v obliki Json). Ukaz S1 lahko ponovimo s klikom na gumb Osveži.

Ukaz	**Smer**	**Dodani podatki**
S1	Uporabniški vmesnik – ESP	/
S1	ESP – Uporabniški vmesnik	Podatki o vozlišču
W1	Uporabniški vmesnik – ESP	/
W1	ESP – Uporabniški vmesnik	Seznam najdenih omrežij
W2	Uporabniški vmesnik – ESP	Nove nastavitve Wi-Fi
A1	Uporabniški vmesnik – ESP	Nove dodatne nastavitve

Tabela 5.1: Seznam vseh ukazov za komunikacijo med uporabniškim vmesnikom in modulom ESP

Vsi ukazi so poslani z uporabnikove strani na modul ESP prek spletnega vtičnika (angl. websocket), nato se v odgovor vrne enak ukaz z dodanimi podatki. Največja težava je bila pri ukazu za shranjevanje novega omrežja

in preklop na novo omrežje v načinu brez ponovnega zagona. ESP je ob prejetem ukazu ustavil trenutno povezavo Wi-Fi in začel z novo povezavo na drugo omrežje, pri tem pa pozabil na aktivne povezave odjemalcev prek spletnega vtičnika na asinhronski strežnik (slika 5.8). Sprožila se je izjema, saj modul ESP ni vedel, kaj narediti z zapadlimi povezavami. Rešitev težave je bila uporaba zastavice in zaprtje spletnega vtičnika (slika 5.10) po izvedbi ukaza za spremembo nastavitev Wi-Fi.

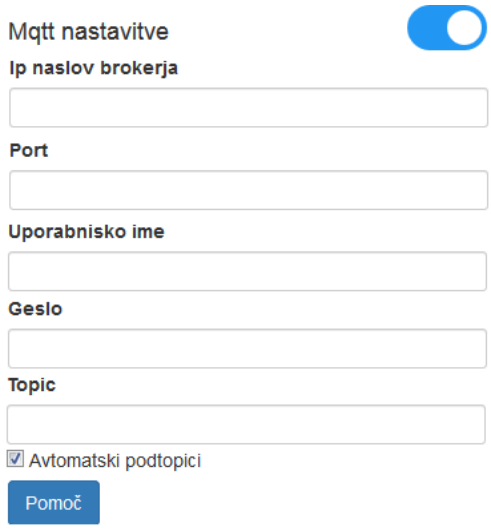
```
var json = {
    'ssid': $('#ssid').val(),
    'passphrase': $('#pass').val(),
    'dhcp': $('.advancedcheckbox').prop('checked'),
    'ip': [parseInt(ip[0]), parseInt(ip[1]), parseInt(ip[2]),
     parseInt(ip[3])],
    'netmask': [parseInt(mask[0]), parseInt(mask[1]),
     parseInt(mask[2]), parseInt(mask[3])],
    'gateway': [parseInt(gateway[0]), parseInt(gateway[1]),
     parseInt(gateway[2]), parseInt(gateway[3])],
    'rebootflag': $('.rebootflag').prop('checked')
};

ws.send('W2'+ JSON.stringify(json));
ws.close();
```

Slika 5.10: Sestava konfiguracije Wi-Fi z uporabniškega vmesnika v objekt Json in zaprtje spletnega vtičnika po poslanem ukazu

Ob poslanem ukazu za nastavitve Wi-Fi z uporabniškega vmesnika, se na modulu ESP dvigne zastavica, ki nakazuje spremembo omrežja Wi-Fi, odjemalec na svoji strani pa po poslanem ukazu avtomatsko zapre spletni vtičnik, in modul ESP lahko z vsemi zaprtimi povezavami z odjemalci začne proces preklopa na drugo omrežje Wi-Fi. Uporabnik ima tudi možnost shraniti nastavitve Wi-Fi z uporabo zastavice za ponovni zagon, po kateri modul ESP shrani nastavitve v pomnilnik in se pred preklopom omrežja znova zažene ter prične povezavo na novo omrežje Wi-Fi. Vse nastavitve, ki pridejo z uporabnikove strani, so takoj zapisane v konfiguracijo modula v pomnilnik.

Protokol Mqtt se na vozlišču zažene, ko uporabnik to želi. Pri tem je po-

trebno možnost Mqtt vključiti in izpolniti vse nastavitve. Modul se poveže na določeno temo (slika 5.11), ki jo uporabnik določi, vendar s prednastavljenimi podtemami, prikazanimi v oknu za pomoč. Če uporabnik želi nastaviti svoje teme za poslušanje ali pošiljanje, lahko s klikom na gumb odpre dodatno okno, kjer lahko vpiše temo za vsako tipalo in relejsko stikalo posebej.



Slika 5.11: Nastavitve protokola Mqtt na vozlišču

5.5 Branje tipal in krmiljenje relejskega stikala

Branje tipal je narejeno s pomočjo prekinitev, saj tako ne ovira in upočasnjuje asinhronskega strežnika Http za uporabniški vmesnik in odjemalca Mqtt. Uporabnik lahko nastavi željen interval branja podatkov v uporabniškem vmesniku. V vsakem ciklu glavne zanke se preveri čas notranje ure ESP-jevega procesorja in ta čas primerja z zadnjim časom branja tipal, ki je shranjen v spremenljivko. Če je razlika med notranjo uro in zadnjim časom branja večja od nastavljenega intervala, se izvede branje tipal. Branje tipala osvetljenosti je opravljeno takoj, temperaturno tipalo DHT pa potrebuje za

branje nekaj časa. Pridobljeni podatki se shranijo v lokalne spremenljivke in pošljejo na strežnik Mqtt. Branje tipala gibanja poteka ločeno in skripta v vsakem ciklu glavne zanke preveri, ali je bilo gibanje zaznano, ter to pošlje na Mqtt. Relejsko stikalo je vezano na temo, ki jo uporabnik določi. Ob spremembi le-te se izvrši sprememba na releju, ampak le če je novo prejeto stanje drugačno od prejšnjega.

5.5.1 Odjemalec Mqtt

Za komunikacijo s strežnikom Mqtt je uporabljena javno dostopna odprtokodna knjižnica PubSubClient [11], ki deluje na protokolu TCP. Na modulu ESP je potrebno deklarirati nov objekt protokola Wi-Fi in na njem odpreti nov PubSubClient s podatki za prijavo, ki so shranjeni v konfiguraciji vozlišča (slika 5.12). ESP se nato poveže na uporabniško določene teme ter na njih posluša. Ob vsaki spremembi teme se pokliče podprogram, ki pridobi podatke o temi in njeni morebitni spremembi.

```
WiFiClient espClient;
PubSubClient client(espClient);

IPAddress mqttserveraa(config.mqttserver);
client.setServer(mqttserveraa, config.mqttport);
client.setCallback(callback);

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Povezujem na mqtt server");
    // Attempt to connect
    if (client.connect(config.device_name.c_str(),config.mqttusername.c_str(),config.mqttpass.c_str())) {
      Serial.println("Povezan");
      client.subscribe(config.mqtttopicrelay.c_str());
    } else {
      Serial.print("Povezava neuspesna");
      Serial.print(client.state());
      break;
    }
  }
}
```

Slika 5.12: Deklaracija objektov in povezava na strežnik Mqtt

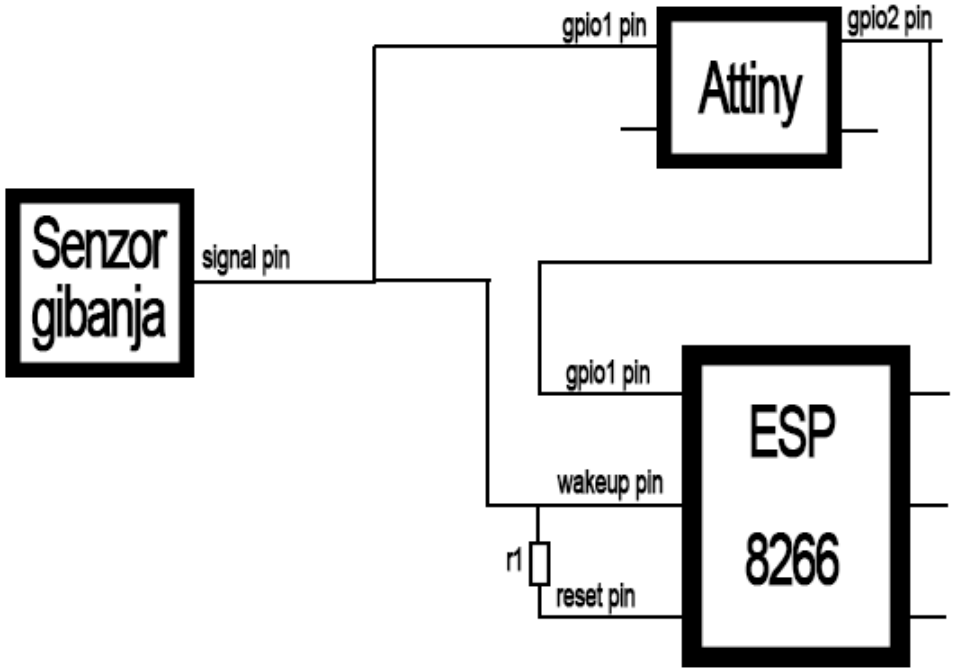
Pri vozlišču na določeno temo posluša samo relejsko stikalo. Vsa ostala tipala objavljajo podatke na teme, določene s strani uporabnika. Za relejsko stikalo je uporabljena dodatna zaščita, katere naloga je preprečiti nezaželjen vklop zaradi spremembe v drugi temi, na katero relejsko stikalo ni naročeno. Če se ime teme, ki je bila spremenjena, ujema z imenom teme, na katero je relejsko stikalo naročeno, in novo stanje drugačno od prejšnjega, se sprememba stanja izvrši.

5.6 Način globokega spanja

Ob priklopu na baterije je priporočeno vklopiti način globokega spanja in nastaviti njegov časovni interval. Vozlišče se po tem intervalu zbudi, poveže na omrežje Wi-Fi in strežnik Mqtt, ter objavi trenutne meritve. V načinu globokega spanja je omogočeno le branje podatkov s tipala temperature, vlage in osvetljenosti. Relejsko stikalo je tako onemogočeno, saj sprememba teme na strežniku med globokim spanjem ne pride do modula ESP. Enaka težava je s tipalom gibanja, ki bi moralo ob zaznanem gibanju modul ESP prebuditi s spanja in sporočiti zaznavo gibanja na strežnik Mqtt. Rešitev za tipalo gibanja smo načrtovali, vendar ni vključena v zadnjo različico vezja vozlišča.

Globoko spanje prebudi signal, posredovan iz priključka za bujenje na modulu ESP, na priključek za ponovni zagon. Signal tipala gibanja bi lahko prebudil modul ESP, vendar mora ESP vedeti, ali ga je prebudil njegov signal za bujenje ali tipalo za gibanje. Rešitev bi bila vključitev še enega mikrokrmilnika (ATtiny) z zelo nizko porabo v vezje. Tako bi ob zaznanem gibanju signal s tipala za gibanje šel naravnost na priključek za ponovni zagon, isti signal pa bi šel še na priključek mikrokrmilnika Attiny, ki bi nato na drugem izhodu poslal signal na drug priključek modula ESP (glej sliko 5.13). Ob ponovnem zagonu modula ESP bi ta preveril, ali je signal na priključku, povezanim z ATtinyijem, visok, kar bi pomenilo, da je za bujenje poskrbelo tipalo gibanja, ali pa je nizek, kar bi pomenilo, da je za bujenje

poskrbela notranja ura globokega spanja s priključka za bujenje. Žal signal s tipala gibanja ne ostane dovolj časa v visokem stanju ob zaznanem gibanju, da bi ga lahko povezali še na priključek modula ESP.



Slika 5.13: Možen popravek za delovanje tipala gibanja tudi pri načinu globokega spanja

Poglavje 6

Prikaz delovanja vozlišča v praksi

V poglavju je prikazana uporaba vozlišča. Vozlišče je potrebno povezati v omrežje ter nastaviti vse potrebne parametre. Če želimo podatke pošiljati na strežnik Mqtt, je potrebno v nastavitvah to omogočiti. Za povezavo z osrednjim sistemom Home Assistant (pogl. 4.2) moramo dodati nekaj dodatnih vrstic v nastavitvah zgoraj omenjenega osrednjega sistema.

6.1 Povezava v omrežje

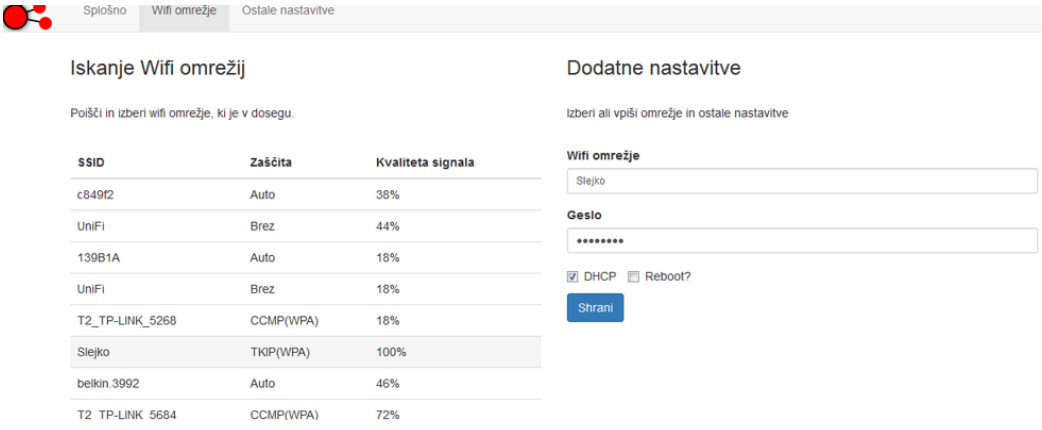
V ESP vključimo priključek USB-mini z napajanjem. Zaradi prazne konfiguracije se nam kmalu po vklopu med omrežji Wi-Fi prikaže omrežje vozlišča. Nanj se povežemo in dostopamo do uporabniškega vmesnika na naslovu 192.168.1.1 prek spletnega brskalnika (slika 6.1).

Nato kliknemo zavihek omrežje Wi-Fi in poiščemo omrežja, ki so nam na voljo s klikom na gumb Išči. ESP nam vrne seznam najdenih omrežji, na katerem si s klikom izberemo omrežje, ki nam ga nato avtomatsko vpiše v vnosno okno za omrežje (slika 6.2). Potreben je še vnos gesla in klik na gumb Shrani. Če želimo, lahko odkljukamo možnost avtomatskega naslova IP in vpišemo svojega. Zraven je še možnost ponovnega zagona po shranjenih

Slika 6.1: Povezava na vozlišče in splošne informacije

nastavitvah. Po kliku na gumb Shrani se vozlišče poveže na določeno omrežje.

Če želimo dostopati do uporabniškega vmesnika, se je potrebno povezati na isto omrežje in poiskati naslov IP vozlišča, kjer bomo nastavili nastavitve Mqtt.

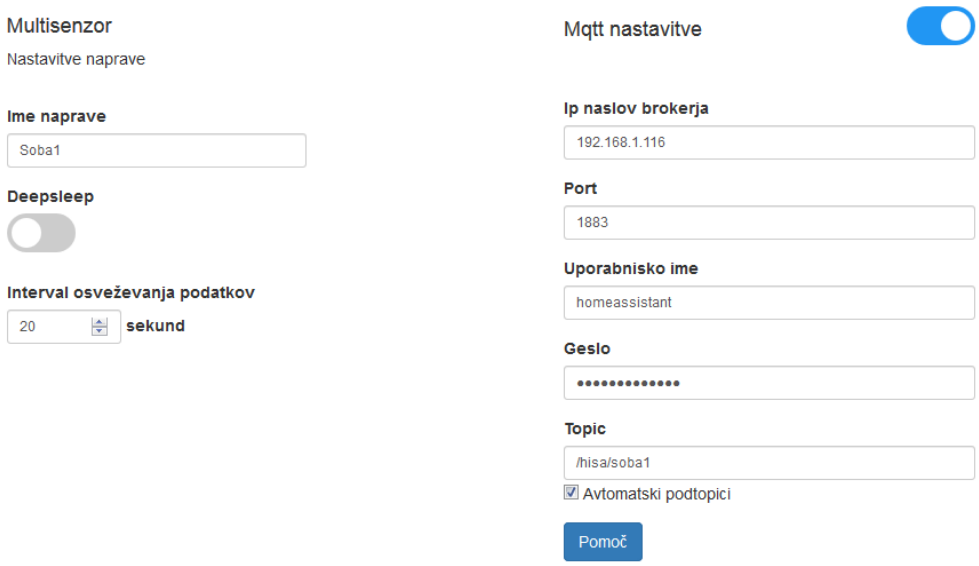


Slika 6.2: Izbira omrežja in povezava nanj

6.2 Nastavitve in povezava po protokolu Mqtt

V tem konkretnem primeru se je vozlišče nahajalo na naslovu IP 192.168.1.126, določen s strani usmerjevalnika. Potrebno je nastaviti še povezavo na strežnik

Mqtt, ki se nahaja na Raspberry PI, dodati vozlišče v konfiguracijo sistema Home Assistant ter pravilno nastaviti teme Mqtt. Ko smo povezani na uporabniški vmesnik, izberemo zavihek Dodatne nastavitve (slika 6.3), vpišemo ime vozlišča (npr. Soba1) in vnesemo interval osveževanja v sekundah. Če želimo uporabiti Mqtt, moramo z drsnikom vklopiti Mqtt, vnesti vse podatke o strežniku, uporabniško ime in geslo za povezavo na strežnik in teme. Pri temah imamo možnost glavne teme, ki so ji nato avtomatsko dodane podteme, te pa najdemo v gumbu za pomoč. Izbrali smo avtomatsko podtemo in vnesli glavno temo sobe */hisa/soba1/*. Po shranjenih nastavitvah se vozlišče poveže na strežnik Mqtt in začne pošiljati podatke. Spremembe so na strežniku Mqtt takoj vidne, če pa jih želimo videti in kontrolirati še v sistemu Home Assistant, moramo dodati nekaj dodatnih vrstic konfiguracije.

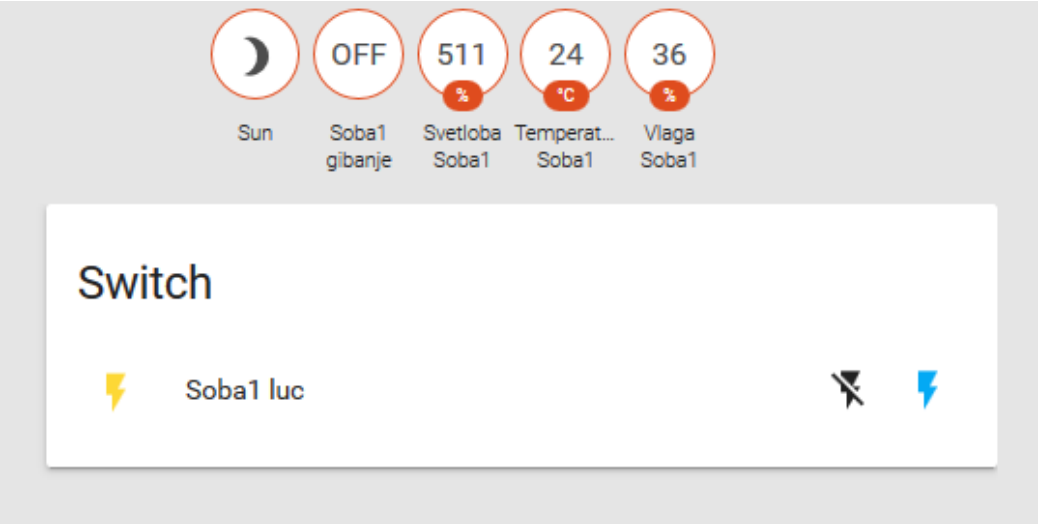


Slika 6.3: Nastavitve Mqtt in ostali parametri

6.3 Povezava z osrednjim sistemom

V sistemu Home Assistant je potrebno v konfiguracijo dodati tipala, teme Mqtt, na katerih delujejo, in relejsko stikalo. Na sliki 6.4 je viden priklop enega od vozlišč na sistem, v realnosti pa je mišljeno v vsak prostor namestiti svoje vozlišče in tako pridobiti brezžično omrežje vozlišč ter nadzor nad celotno hišo z vseh pametnih naprav. Spodaj je primer, kako dodamo tipalo temperature v konfiguracijo.

```
sensor:
  - platform: mqtt
    state_topic: '/hisa/soba1/temp/'
    name: 'Temperatura Soba1'
    unit_of_measurement: '°C'
```



Slika 6.4: Prikaz vozlišča v programju Home Assistant

Poglavje 7

Sklepne ugotovitve

V okviru diplomskega dela je predstavljen celotni razvoj ideje o brezžičnem vozlišču, ki vsebuje tipala in aktuator. Dela je bilo veliko, saj se je bilo potrebno naučiti še osnov načrtovanja vezij in ugotoviti, kako nekatere električne komponente delujejo, hkrati pa upoštevati želje potencialnega uporabnika. Vezje je bilo kar nekajkrat spremenjeno, saj se je ves čas ciljalo na začetno idejo po preprostem vezju, ki ga lahko vključimo in nastavimo brez posebnega predhodnega znanja. Pri prvi različici vezja je bilo veliko težav zaradi začetniških napak pri spajkanju majhnih komponent na vezje. Po izdelavi druge različice vezja se je večino napak odpravilo in testiranja so pokazala napredek. Pri testiranjih nas je presenetil modul ESP8266, saj za to majhno ceno ponuja res veliko in smo mnenja, da bi se ga dalo uporabiti pri veliko zanimivih projektih.

Razvoj programske opreme je predstavljal še večji izziv – to je namreč stik med uporabnikom in vozliščem. Že na začetku programiranja si je bilo potrebno izdelati zelo dober diagram poteka in v njem zastaviti čim več možnih stanj in odločitev. Med razvojem programske opreme so se pojavljale napake in težave. Največje so bile tiste s preklopom omrežja Wi-Fi, saj nam modul ESP ni vračal nobenih informacij, zakaj se programska oprema ustavi. Po nekaj dnevih razmišljanja smo celoten algoritem strežnika pregledali od vrha do dna in ugotovili, v čem je težava. Kot je pri reševanju programerskih težav

običajno, se je za najbolj logično rešitev izkazala najpreprostejša. Težava je bila v odprtih povezavah med uporabnikom in modulom ESP, katere so se po preklopu omrežja izgubile in modul ESP ni vedel kam z njimi. Na podlagi tega smo spoznali, da je potrebno zapreti vse povezave, ki jih ne potrebujemo več. Pri načrtovanju uporabniškega vmesnika je bilo potrebno paziti na dinamičnost spletne aplikacije, saj si vsak uporabnik želi dostop do sistema z vseh možnih pametnih naprav, ki jih ima doma.

Izboljšav pri takem izdelku je lahko veliko. Sistem je bil načrtovan modularno z možnostjo nadgradnje. Razvili bi še dodatna vezja, ki se lahko priključijo na vozlišče, in s katerimi lahko uporabnik doda nove možnosti delovanja. Če bi si uporabnik želel z vozliščem krmiliti še več zunanjih naprav, bi lahko zgolj dodal vezje z več relejskimi stikali in ga povezal z vozliščem. Na tak način lahko pridemo do sistema, ki se modularno nadgrajuje in uporablja po želji. Izboljšave bi naredili tudi v smeri baterijskega napajanja, saj nimamo povsod dovoda električne energije.

Izdelek se je testiral znotraj objekta in rezultati so bili zelo dobri. Hitra odzivnost, zanesljivost pri delovanju in stabilnost povezave. Ljudje potrebujemo zanesljive in varne sisteme z enostavno možnostjo nadgradnje in čim manj poseganja v obstoječi objekt. To diplomsko delo je vsekakor koristna izkušnja in zelo obetavno izhodišče za nadaljnji razvoj takega sistema.

Literatura

[1] "Altum Designer", orodje za načrtovanje vezij. Dosegljivo: http://www.altium.com/altium-designer/. [Dostopano 20. 11. 2017].

[2] "Arduino IDE", razvojno okolje. Dosegljivo: https://www.arduino.cc/en/Main/Software. [Dostopano 5. 12. 2017].

[3] "Bootstrap", platforma za oblikovanje spletnih aplikacij. Dosegljivo: https://getbootstrap.com/. [Dostopano 10. 12. 2017].

[4] Delovanje tipala DHT. Dosegljivo: https://learn.adafruit.com/dht/overview. [Dostopano 10. 11. 2017].

[5] Delovanje tipala gibanja. Dosegljivo: https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/how-pirs-work. [Dostopano 10. 11. 2017].

[6] "ESPAsyncWebServer", knjižnica asinhronskega spletnega strežnika. Dosegljivo: https://github.com/me-no-dev/ESPAsyncWebServer. [Dostopano 7. 12. 2017].

[7] "Home Assistant", program za hišno avtomatizacijo. Dosegljivo: https://home-assistant.io/. [Dostopano 2. 12. 2017].

[8] "Json", knjižnica za pretvorbo v obliko Json. Dosegljivo: https://github.com/bblanchon/ArduinoJson. [Dostopano 7. 12. 2017].

[9] Mqtt odjemalec, različica Mosquitto. Dosegljivo: https://mosquitto.org/. [Dostopano 16. 12. 2017].

[10] Protokol Mqtt. Dosegljivo: http://mqtt.org/faq. [Dostopano 1. 12. 2017].

[11] "Pubsub client", knjižnica za uporabo Mqtt protokola. Dosegljivo: https://github.com/knolleary/pubsubclient. [Dostopano 7. 12. 2017].

[12] Slika delovanja Brokerja Mqtt. Dosegljivo: https://pagefault.blog/2017/03/02/using-local-mqtt-broker-for-cloud-and-interprocess-communication/. [Dostopano 16. 12. 2017].

[13] Slika modula ESP12e. Dosegljivo: https://cdn3.volusion.com/btfzd.umflq/v/vspfiles/photos/AD247-2.jpg?1456512809. [Dostopano 10. 11. 2017].

[14] Slika računalnika Raspberry PI B+. Dosegljivo: https://www.raspberrypi.org/products/model-b-plus/. [Dostopano 10. 11. 2017].

[15] Slika tipala DHT11. Dosegljivo: https://www.hellasdigital.gr/images/detailed/5/dht11.jpg. [Dostopano 10. 11. 2017].

[16] Slika uporabniškega vmesnika programa Home Assistant. Dosegljivo: https://www.mysensors.org/uploads/57be15b86b0aea1b61746265/394/homeassistant_devices.png. [Dostopano 2. 12. 2017].

[17] "SPIFFS", knjižnica za dostop do pomnilnika. Dosegljivo: http://esp8266.github.io/Arduino/versions/2.0.0/doc/filesystem.html. [Dostopano 7. 12. 2017].

[18] Varnost protokola Mqtt. Dosegljivo: https://dzone.com/articles/mqtt-security-securing-a-mosquitto-server. [Dostopano 1. 12. 2017].

[19] Raspberry Pi Foundation. Računalnik Raspberry PI B+. Dosegljivo: `https://www.raspberrypi.org/products/raspberry-pi-1-model-b/`. [Dostopano 10. 11. 2017].

[20] Drew Hendricks. Zgodovina pametnih hiš. Dosegljivo: `http://www.iotevolutionworld.com/m2m/articles/376816-history-smart-homes.htm`, 2014. [Dostopano 5. 1. 2018].

[21] OASIS Open. Protokol Mqtt, različica 3.1.1. Dosegljivo: `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf`, 29. [Dostopano 1. 12. 2017].

[22] Dag Spicer. The ECHO IV Home Computer: 50 Years Later. Dosegljivo: `http://www.computerhistory.org/atchm/the-echo-iv-home-computer-50-years-later/`, 2016. [Dostopano 5. 1. 2018].

[23] Espressif Systems IOT Team. Mikrokrmilnik ESP8266. Dosegljivo: `https://cdn-shop.adafruit.com/product-files/2471/0A-ESP8266__Datasheet__EN_v4.3.pdf`, 2015. [Dostopano 10. 11. 2017].