Programska implementacija sustava udaljene kontrole i nadzora uređaja baziranih na Arduino platformi

Grobenski, Jože

Undergraduate thesis / Završni rad

2015

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University North / Sveučilište Sjever**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:122:774539

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-07-16**

Repository / Repozitorij:

University North Digital Repository

zir.nsk.hr

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

Sveučilište
Sjever

Završni rad br. 326/EL/2015

Programska implementacija sustava udaljene kontrole i nadzora uređaja baziranih na Arduino platformi

Jože Grobenski, 4121/601

Varaždin, rujan 2015.

Sveučilište
Sjever

Odjel za elektrotehniku

Završni rad br. 326/EL/2015

Programska implementacija sustava udaljene kontrole i nadzora uređaja baziranih na Arduino platformi

Student

Jože Grobenski, 4121/601

Mentor

mr. sc. Matija Mikac, viši predavač

Varaždin, rujan 2015.

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

UNIVERSITY NORTH

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL: Odjel za elektrotehniku

PRISTUPNIK: Jože Grobenski | MATIČNI BROJ: 4121/601

DATUM: 09.07.2015

KOLEGIJ: Programski jezici i algoritmi

NASLOV RADA: Programska implementacija sustava udaljene kontrole i nadzora uređaja baziranih na Arduino platformi

MENTOR: Matija Mikac | ZVANJE: viši predavač

ČLANOVI POVJERENSTVA:
1. Ladislav Havaš
2. Ivan Šumiga
3. Matija Mikac

Zadatak završnog rada

BROJ: 362/EL/2015

OPIS

Završnim radom je potrebno razmotriti mogućnosti udaljene kontrole i nadzora uređaja baziranih na Arduino platformi - obzirom da je to vrlo popularna i rasprostranjena platforma kojom je relativno jednostavno realizirati različite funkcionalnosti, vrlo je korisna mogućnost iskorištavanja otvorenosti platforme. Upravo je to predmet istraživanja u ovom radu. Same funkcionalnosti uređaja koji će se koristiti za testiranje nisu bitne - dovoljno je implementirati jednostavni uređaj koji posjeduje nekoliko izlaza (npr. LED diode, sklopke i slično). Bitno je realizirati mogućnost kontroliranja i nadzora (očitanje stanja) tih izlaza!

U radu je potrebno:

* dati kratki osvrt na Arduino platformu
* proučiti i dokumentirati mogućnosti udaljene kontrole i nadzora, uz pregled potrebne opreme i modula za realizaciju
* realizirati jednostavni uređaj baziran na Arduino platformi koji će uključivati nekoliko kontrolabilnih izlaza
* osmisliti i implementirati (programski, na Arduino uređaju i na mobilnom uređaju) mogućnost kontrole i nadzora uređaja korištenjem Bluetooth i/ili IrDA komunikacije, uz implementaciju kontrolne aplikacije na mobilnom uređaju (Android, Windows Phone...)
  * realizaciju mobilne aplikacije izvesti korištenjem pojednostavljenog razvojnog sustava AppInventor
  * realizaciju mobilne aplikacije izvesti korištenjem standardnog razvojnog sustava Android Studio
  * usporediti postupke realizacije i dati komentare na mogućnosti spomenutih razvojnih sustava
* osmisliti i implementirati mogućnosti udaljene kontrole i nadzora uređaja preko web sučelja javno dostupnog poslužitelja
  * realizaciju izvesti korištenjem web poslužitelja i potrebnih modula na samom Arduino uređaju
  * realizaciju izvesti korištenjem javnog poslužitelja na kojeg Arduino šalje i s kojeg prima informacije
  * usporediti postupke realizacije, komentirati mogućnosti i eventualne razlike u funkcioniranju (npr. odziv i slično)
* rekapitulirati troškove razvoja Arduino sustava - jezgra i potrebni moduli
* ukazati na mogućnosti korištenja projektiranog sustava u praksi (primjena)

ZADATAK URUČEN: 15. 09. 2015. | POTPIS MENTORA: M. Mikac

SVEUČILIŠTE SJEVER

Predgovor

Najiskrenije zahvale mentoru mr. sc. Matiji Mikacu, za dijeljenje znanja i iskustva te uloženo strpljenje i trud kroz proces nastajanja ovog rada. Ovim radom sam stekao iskustvo i znanje koje će koristiti u mom daljnjem razvoju, kao akademski obrazovane osobe. Također, hvala Sveučilištu, profesorima, asistentima, kolegi Matiji Budenu i svima koji su mi davali korisne savjete i pružili pomoć prilikom pisanja rada.

Sažetak

Ovim radom prikazana je mogućnost udaljene kontrole i nadzora uređaja baziranog na *Arduino* platformi. Obzirom da je riječ o vrlo popularnoj i rasprostranjenoj platformi, koja nudi različite funkcionalnosti, upravo je ona predmet istraživanja rada. Naime, implementiran je jednostavan uređaj koji posjeduje nekoliko izlaza (*LED* diode) te je realizirana mogućnost kontroliranja i nadzora tih izlaza na istome. Drugim riječima, realizirana je (na *Arduinu* i mobilnom uređaju) mogućnost kontrole i nadzora uređaja korištenjem *Bluetooth* bežične tehnologije (idealne za manje udaljenosti na svega desetak metara), uz implementaciju kontrolne aplikacije na mobilnom uređaju (*Android* operacijski sustav). Taj je dio realiziran na dva načina: korištenjem pojednostavljenog razvojnog sustava *App Inventor* te korištenjem standardnog razvojnog sustava *Android Studio*. S druge strane, implementirana je mogućnost udaljenje kontrole i nadzora uređaja preko web sučelja javno dostupnog poslužitelja, također u dvije izvedbe. Prva je korištenjem web poslužitelja i potrebnih modula na samoj *Arduino* platformi, a druga je korištenjem javnog poslužitelja na kojeg se *Arduino* spaja kao klijent, šalje i s kojeg čita informacije.

Popis korištenih kratica

AC	Alternating Current
ADT	Android Development Tools
API	Application Programming Interface
CSS	Cascading Style Sheets
DC	Direct Current
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
EDR	Enhanced Data Rate
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IP	Internet Protocol
ISM	Industrial, Scientific and Medical
LED	Light Emitting Diode
MAC	Media Acces Control
MISO	Master In Slave Out
MOSI	Master Out Slave In
NAT	Network Address Translation
PAT	Port Address Translation
PHP	HyperText Preprocessor
POE	Power over Ethernet
SCK	Serial Clock
SD	Secure Digital
SDK	Software Development Kit
SPI	Serial Peripheral Interface
SPP	Serial Port Profile
SS	Slave Select
TCP	Transmission Control Protocol
TTL	Transistor Transistor Logic
UDP	User Datagram Protocol
URL	Uniform Resource Locator
UTP	Unshielded Twisted Pair
XML	Xtensible Markup Language

Sadržaj

1. UVOD ... 1

2. O ARDUINO PLATFORMI ... 2

2.1. Arduino razvojna okolina ... 3

3. IMPLEMENTACIJA UREĐAJA ... 5

3.1. Mjerenje napona ... 7

4. KONTROLA PREKO BLUETOOTH BEŽIČNE TEHNOLOGIJE ... 9

4.1. Android operacijski sustav ... 9

4.2. Bluetooth bežična tehnologija ... 9

4.2.1. Bluetooth modul HC-06 ... 10

4.3. Protokol komunikacije ... 13

4.4. Opis Arduino programa za aplikacije ... 14

4.5. App Inventor ... 20

4.5.1. Princip rada aplikacije u App Inventoru ... 21

4.5.2. Logički dio aplikacije u App Inventoru ... 22

4.6. Android Studio ... 27

4.6.1. Opis programskog koda izrađene aplikacije u Android Studiu ... 28

4.7. Usporedba razvojnih sustava App Inventor i Android Studio ... 35

5. KONTROLA PREKO WEB SUČELJA JAVNO DOSTUPNOG POSLUŽITELJA ... 36

5.1. Ethernet modul ... 36

5.2. Arduino kao poslužitelj ... 37

5.2.1. Princip rada Arduino poslužitelja ... 38

5.2.2. Web stranica ... 40

5.2.3. Pristup poslužitelju sa javne mreže (Internet) ... 40

5.3. Arduino kao klijent ... 43

5.3.1. PHP programski jezik ... 44

5.3.2. Funkcije pojedinih dijelova koda ... 44

5.4. Usporedba Arduina u ulozi poslužitelja i u ulozi klijenta ... 50

6. OPTIMIZACIJA PROJEKTIRANOG SUSTAVA ... 51

6.1. Optimizacija prometa putem Bluetooth komunikacije ... 51

6.2. Optimizacija kad je Arduino poslužitelj ... 51

6.3. Optimizacija kad je Arduino klijent ... 53

7. REKAPITULACIJA TROŠKOVA ... 55

8. PRIMJENA REALIZIRANOG SUSTAVA U PRAKSI ... 56

9. ZAKLJUČAK ... 58

10. LITERATURA ... 59

11. POPIS SLIKA I TABELA ... 60

12. PRILOZI ... 61

1. Uvod

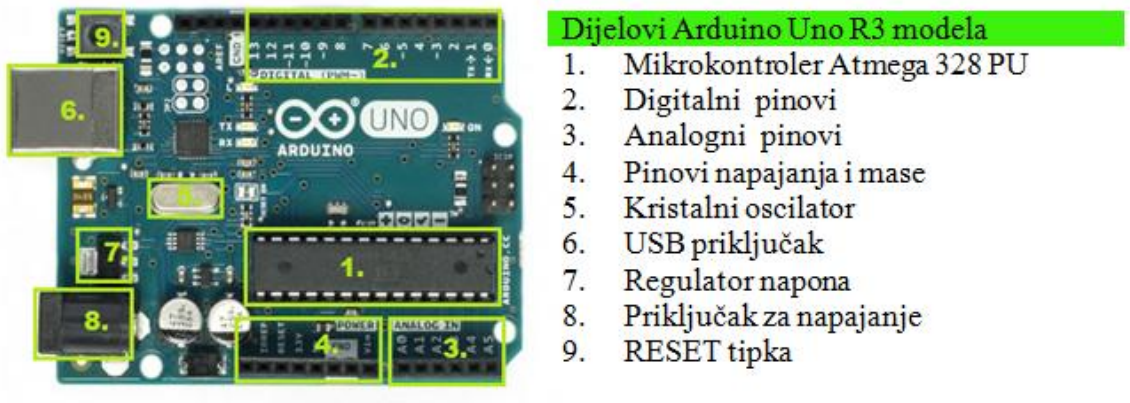
U radu je bilo potrebno realizirati jednostavan uređaj te omogućiti udaljenu kontrolu i nadzor tog uređaja. Jezgru rada predstavlja *Arduino* platforma, gdje programski dio iste čini besplatna razvojna okolina. Nadalje, korištena su dva načina udaljene kontrole. Jedan je preko *Bluetooth* bežične komunikacije, a drugi putem web sučelja (web stranice), tj. javno dostupnog poslužitelja te je idealan za globalno upravljanje (Internet). Za *Bluetooth* kontrolu izrađene su mobilne aplikacije za *Android* operativni sustav u pojednostavljenom razvojnom okruženju *App Inventor* i u standardnom razvojnom okruženju *Android Studio*. Činjenica je da je *App Inventor* jednostavan, namijenjen svima, dok je *Android Studio* ipak nešto složeniji te zahtijeva poznavanje *Java* programskog jezika. Aplikacije i *Arduino* nemaju direktnu vezu već oni komuniciraju preko *Bluetooth HC-06* modula koji služi kao posrednik u komunikaciji.

U drugom dijelu, tj. za kontrolu preko web sučelja napravljene su dvije izvedbe. Kod prve izvedbe koristi se web poslužitelj na samom *Arduinu* te se programski generira web stranica za upravljanje, dok se kod druge varijante izvedbe upravljanje i nadzor realizira preko web stranice na javnom poslužitelju s kojim *Arduino*, u ulozi klijenta, komunicira, čita i prosljeđuje potrebne informacije. Za realizaciju udaljene kontrole uređaja preko web stranice korišten je *Ethernet Shield*, odnosno modul koji *Arduinu* omogućuje povezivanja na lokalnu i javnu mrežu. Iz svega se na kraju zaključuje da je realizacija *Arduina* kao poslužitelja idealna za manje i ne zahtjevne projekte upravljanja. S druge strane, izvedba kad je *Arduino* klijent, ističe prednosti. Naime, tu *Arduino* samo šalje i čita podatke sa poslužitelja. Iz toga se zaključuje kako se veći dio posla obavlja na poslužitelju.

2. O Arduino platformi

Arduino platforma je platforma otvorenog koda koja se koristi za izradu raznih projekata iz područja elektronike, automatizacije i slično. Sastoji se od fizičkog dijela i programskog, odnosno softverskog (eng. *software*) dijela. Fizički dio čini tiskana pločica na kojoj se nalazi najbitniji dio same *Arduino* platforme, odnosno programirljivi mikrokontroler, dok programski dio čini integrirano razvojno okruženje, tzv. *IDE* (eng. *Integrated Development Environment*). Postoji nekoliko različitih modela *Arduino* platforme, a model koji je korišten u ovom radu je *ArduinoUno R3*. Ovaj model baziran je na 8 bitnom mikrokontroleru *Atmega 328P*.

Radni napon *Arduino* pločice iznosi 5 V. Dozvoljeni napon na koji se može priključiti iznosi od 6 do 20 V DC, a preporučeni iznosi od 7 do 12 V DC. Ukupno ima 14 digitalnih i 6 analognih pinova. Pinovi mogu biti konfigurirani kao ulazni ili izlazni. Analogni pinovi spojeni su na analogno-digitalni pretvornik (10 bitni). Tako se, primjerice analogna vrijednost napona u rasponu 0-5 V pomoću analogno-digitalnog pretvornika pretvara u digitalnu vrijednost u rasponu od 0 do 1023, odnosno ima 1024 vrijednosti (2^{10} = 1024). Takt kristalnog oscilatora iznosi 16 MHz, dok je maksimalna struja po pinu 40 mA. Digitalni pinovi mogu se nalaziti u dva stanja; logičkoj jedinici (5 V) i logičkoj nuli (0 V). Kod digitalnih pinova vrijedi posebno spomenuti pinove *Rx* (0) i *Tx*(1). *Rx* (eng. *receive*) pin služi za primanje podataka, dok *Tx* (eng. *transmit*) pin služi za slanje podataka. Preko tih pinova odvija se serijska komunikacija. Serijskom komunikacijom preko *USB* kabla program se sa računala prenosi na mikrokontroler. Između ostalog, preko tog kabla *Arduino* može dobivati napajanje.

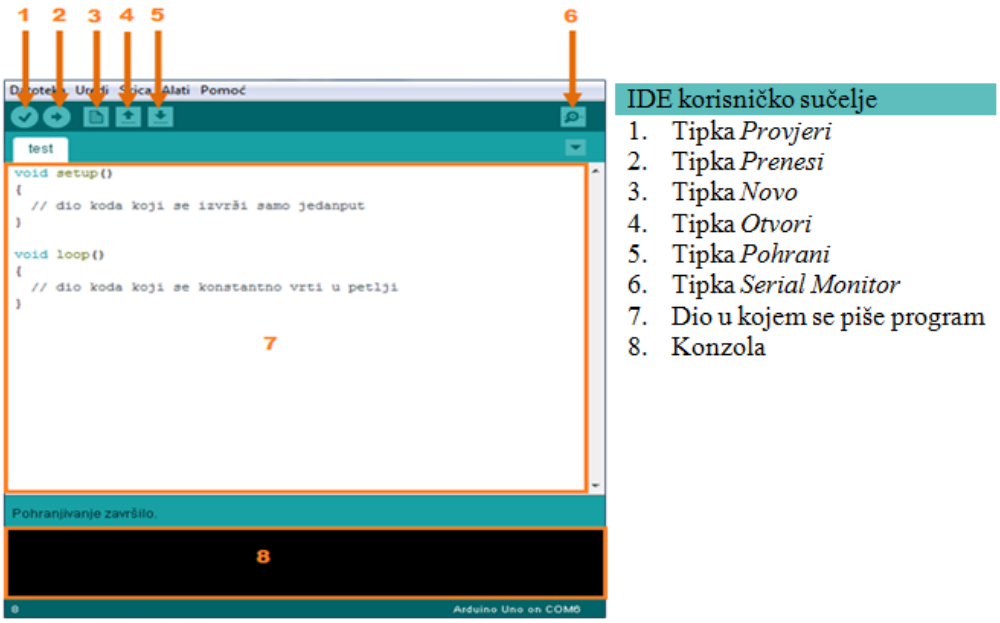


Slika 2-1. Arduino Uno R3 model i najbitniji dijelovi

2.1. Arduino razvojna okolina

Programski dio *Arduino* platforme čini razvojna okolina koja se može besplatno preuzeti na web stranici [1]. U njoj se piše program, tzv. *sketch* koji se kasnije prenosi na mikrokontroler kako bi mikrokontroler obavljao određenu funkciju koja mu je definirana u samom programu. Programski jezik kojim se pišu programi za *Arduino* platformu temeljen je na C/C++ programskom jeziku. Struktura koja čini *sketch* sastoji se od dva dijela, odnosno funkcije. Prva funkcija naziva se *setup* i poziva se samo jedanput, kod početka izvršavanja programa ili kod ponovnog pokretanja (RESET-a) mikrokontrolera. U njoj se, primjerice, definiraju ulazni, izlazni pinovi, deklariraju se i inicijaliziraju varijable i sl. Drugi dio *sketch-a* čini funkcija *loop* i, kao što samo ime govori, kod unutar ovog dijela izvodi se u „beskonačnost".

Program napisan u razvojnom okruženju najprije treba proći provjeru ispravno napisane sintakse. Nakon što je provjera uspješno izvršena, odnosno sintaksa dobro napisana, program se kompajlira (proces pretvaranja jezika razumljiv čovjeku, u instrukcije razumljive mikrokontroleru, tzv. strojni jezik) i nakon toga se može prenijeti na mikrokontroler. Prijenos se obavlja pritiskom na tipku *Prenesi* u njegovom razvojnom okruženju, vidljivo na slici 2-2.



Slika 2-2. Prikaz IDE korisničkog sučelja

Sketchevi su spremljeni sa ekstenzijom .ino. Konzola prikazuje greške i detalje o grešci, ukoliko one postoje. U razvojnom okruženju u donjem desnom kutu nalazi se informacija na kojem serijskom portu računala se nalazi *Arduino* i koji model se koristi. Gumbi u alatnoj traci nam omogućavaju provjeru, prijenos, kreiranje, otvaranje i spremanje *sketcheva* te otvaranje *Serial monitor-a,* a on služi za prikaz podataka koji se šalju ili primaju putem serijske komunikacije.

3. Implementacija uređaja

U radu je bilo potrebno realizirati jednostavan uređaj koji posjeduje nekoliko izlaza. Kao izlazi uređaja korištene su *LED* (eng. *Light Emitting Diode*) diode, pri čemu je cilj bio omogućiti udaljenu kontrolu i nadzor tog uređaja. Prikaz cjelokupne strukture uređaja vidljiv je na slici 3-1. Konkretno, bilo je potrebno realizirati udaljenu kontrolu izlaza (*LED* dioda) uređaja i čitanje stanja tih izlaza. Izlazima je moguće upravljati na dva načina: na samom uređaju pomoću prekidača te programski (preko pinova *Arduina*- pinovi 5, 6 i 7). Izlazima *Arduina* moguće je upravljati na način da dovedemo visoki naponski nivo (5 V) ili niski naponski nivo (0 V). Ti pinovi konfigurirani su kao izlazni, odnosno nalaze se u stanju malog otpora i mogu vanjske krugove opskrbljivati strujom maksimalnog iznosa do 40 mA, što je i više nego dovoljno da bi *LED* dioda svijetlila. Pošto su konfigurirani kao izlazni, u daljnjem tekstu ćemo te pinove *Arduina* nazivati *izlaz1* (pin 5), *izlaz2* (pin 6) i *izlaz3* (pin 7).

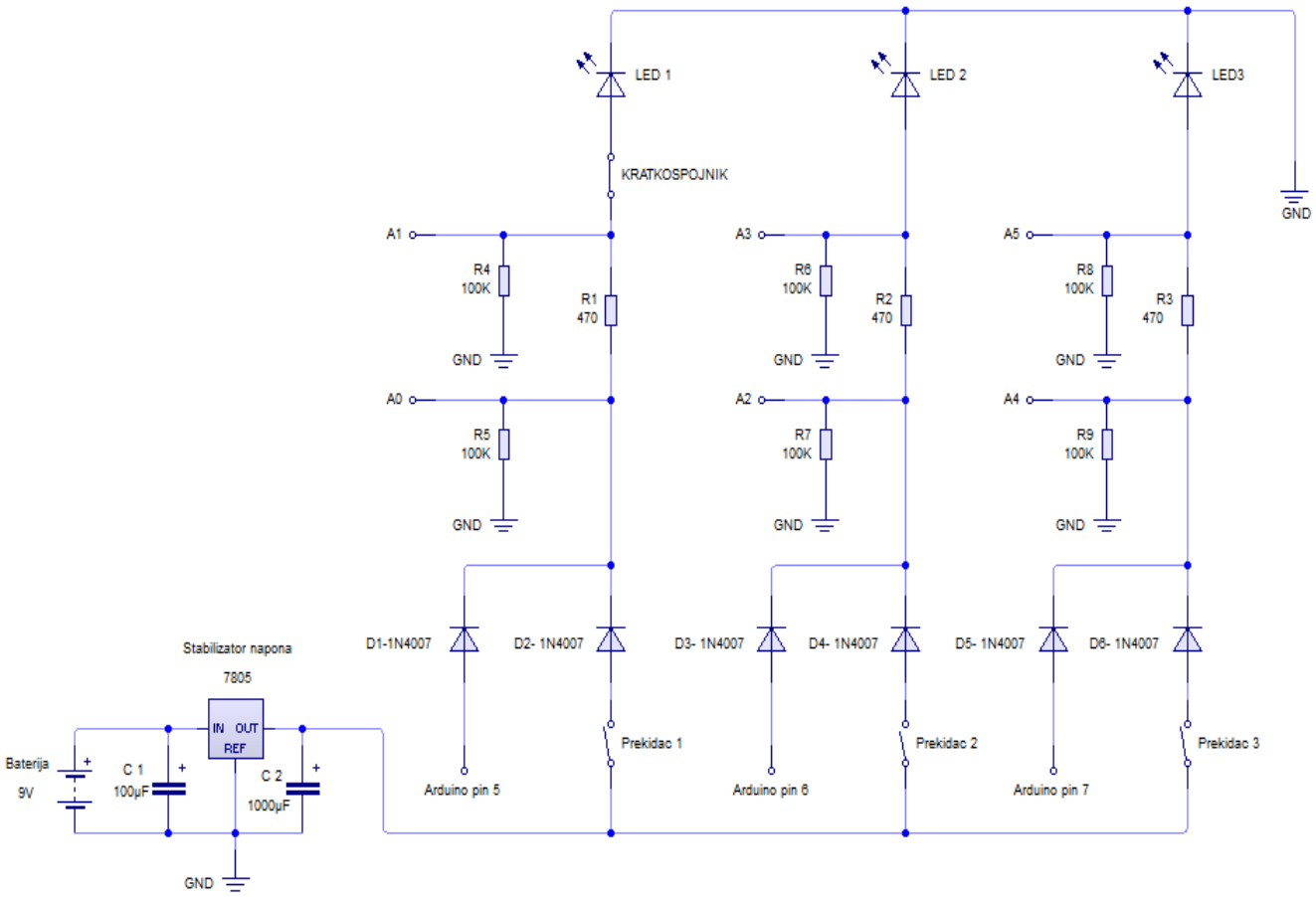
U svrhu zasebnog upravljanja pomoću prekidača, uređaju je dovedeno dodatno napajanje. Napajanje se dobiva iz baterije napona 9 V, a preko stabilizatora napona 7805, napon se regulira, odnosno smanjuje na 5 V. Korišteni stabilizator napona jedan je od predstavnika stabilizatora napona iz serije 78XX. Veličinu izlaznog napona označavaju znamenke XX, što u ovom slučaju iznosi 5 V. Stabilizator napona ima tri izvoda. Na ulazni, krajnje lijevi, izvod se dovodi ulazni napon (9 V), a na izlaznom izvodu (krajnje desni) se dobije stabilizirani napon od 5 V. Izvodi se mogu opteretiti maksimalnom strujom do 1 A. Kod većih opterećenja djeluje unutarnja zaštita (dopušteni ulazni napon je 35 V). Radi smanjenja utjecaja prijelaznih pojava dodaju se paralelno ulazu i izlazu elektrolitski kondenzatori. U ovom slučaju, za ulazni krug korišten je kondenzator kapaciteta 100 µF, a za izlazni krug kondenzator od 1000 µF. Nadalje, diode služe kako struja ne bi tekla u suprotnom smjeru, a koriste se *1N4007* diode kojima zaporni napon iznosi 35 V.

U aplikaciji (i web stranici) je moguće kontrolirati *izlaz1*, *izlaz2* i *izlaz3 Arduina*. Pritom se u aplikaciji prikazuju stanja tih izlaza te se prikazuju stanja samih *LED* dioda. Na taj se način točno zna tko je uključio *LED* diodu (mi programski, ili netko lokalno preko prekidača). Stanje samih izlaza uređaja, odnosno *LED* dioda očitava se na način da se na otporniku (470 Ω) prije *LED* diode mjeri pad napona i na temelju te veličine možemo utvrditi stanje *LED* diode. Ovaj način odabran je iz nekoliko razloga. Kad bi, primjerice, mi programski preko *Arduino* pinova isključili *LED* diodu i stanje te *LED* diode pretpostavljali samo na temelju čitanja stanja tih

pinova *Arduina*, ne bismo nikako mogli znati je li *LED* dioda uključena ili isključena, iz razloga što se ona može upaliti i lokalno na samom uređaju. Također, ukoliko bi je mi programski uključili (npr. preko *izlaza1*) i vidjeli da je *izlaz1* na *Arduinu* uključen, a *LED* dioda je pregorjela, također ne bi dobivali dobre informacije. Prema tome, stanje same *LED* diode ne može se temeljiti na očitanju stanja *izlaza 1,2 i 3 Arduina*. Iz tog se razloga koristi spomenuta metoda mjerenja napona, uz koju sa sigurnošću možemo znati stanje samih *LED* dioda. Naime, ukoliko netko lokalno uključi *LED* diodu preko prekidača, to je vidljivo u aplikaciji na temelju vrijednosti napona. Također, ukoliko se programski želi upaliti *LED* dioda i vidimo da smo postavili *izlaz1* na 5 V, a u aplikaciji vidimo da je stanje *LED*1 ugašeno, znamo da je dioda pregorjela. Simulaciju „pregorjele" diode možemo očitati na *LED* 1 jer je prije nje stavljen kratkospojnik koji nam to omogućuje.

Za mjerenje razlike potencijala, odnosno napona, koristimo analogne pinove od *A0* do *A5*. Pod pretpostavkom kada teče struja, pinovi *A0, A2* i *A4* mjere veći napon, a pinovi *A1, A3* i *A5* niži napon. Oduzimamo vrijednosti (*A0-A1, A2-A3, A4-A5*) i na temelju tih vrijednosti, odnosno razlike potencijala, očitavamo stanje *LED* dioda. Analogni pinovi su konfigurirani kao ulazni i omogućuju mjerenje napona u rasponu od 0 do 5 V. Kako bi umjesto vrijednosti od 0 do 1023 dobivali pravu vrijednost napona, potrebno je veličinu koju mjere analogni pinovi pomnožiti sa (5.0/1023) i na taj način dobivamo vrijednost napona izraženu u voltima.

Obzirom da su analogni pinovi veoma osjetljivi, oni u stanju kad nema napona, mjere, odnosno hvataju bilo kakve smetnje pa tako ponekad vrijednosti koje pinovi mjere prelaze i napone od 1 V, a vrijednosti bi trebale biti 0 V. Takvo mjerenje je neprihvatljivo pa su, u svrhu eliminacije smetnji, sa analognih pinova spojeni otpornici velikog otpora (100 kΩ) na masu, koji svaku smetnju uzemlje. Velik otpor izabran je iz razloga da struja koja njima teče bude gotovo zanemariva te na taj način znatno ne utječu na prilike u strujnom krugu, odnosno mjerenja, a, s druge strane, njihovim dodavanjem smo uvelike smanjili pogreške kod mjerenja napona. U prilogu se nalazi slika implementiranog uređaja (prilog 1), a na slici 3-1. vidi se električna shema uređaja.



Slika 3-1. Električna shema implementiranog uređaja

3.1. Mjerenje napona

Mjerenje napona bez otpornika	
Sve LED diode su isključene i nema dodanih otpornika	Dioda LED1 je upaljena, a ostale su isključene
0.14,0.11,razlika1 je: 0.03	4.39,1.83,razlika1 je: 2.56
0.00,0.00,razlika2 je: 0.00	0.35,0.27,razlika2 je: 0.07
0.53,0.53,razlika3 je: 0.00	0.73,0.70,razlika3 je: 0.02
**********	**********
0.22,0.20,razlika1 je: 0.02	4.39,1.83,razlika1 je: 2.56
0.01,0.00,razlika2 je: 0.01	0.37,0.30,razlika2 je: 0.06
0.63,0.63,razlika3 je: 0.00	0.81,0.80,razlika3 je: 0.01
**********	**********
0.24,0.22,razlika1 je: 0.01	4.39,1.83,razlika1 je: 2.56
0.04,0.02,razlika2 je: 0.02	0.37,0.31,razlika2 je: 0.06
0.71,0.70,razlika3 je: 0.01	0.75,0.75,razlika3 je: 0.00
**********	**********
0.18,0.17,razlika1 je: 0.00	4.39,1.83,razlika1 je: 2.56
0.04,0.03,razlika2 je: 0.01	0.34,0.27,razlika2 je: 0.07
0.69,0.69,razlika3 je: 0.00	0.64,0.62,razlika3 je: 0.01

*Slika 3-2. Mjerenje napona bez otpornika od 100 k*Ω

Na slici 3-2. i slici 3-3. prikazani su rezultati mjerenja napona i usporedba rezultata bez i sa dodanim otporima (100 kΩ), tj. slike prikazuju četiri uzastopna mjerenja razlike napona na otpornicima *R1, R2 i R3*. *Razlika1* prikazuje pad napona na otporniku *R1*, *razlika2* na otporniku *R2* i *razlika 3* na otporniku *R3*. Vrijednosti prije razlike, odvojene zarezom, prikazuju napon na svakom kraju otpornika (*R1, R2, R3*).

Drugim riječima, lijeva strana na slici 3-2. prikazuje mjerenje kad su sve *LED* diode isključene i bez dodanih otpornika (100 kΩ). Iako je razlika napona gotovo 0V, kao što bi i trebala biti, vrijednosti napona koje *Arduino* mjeri na krajevima otpornika su velike, a ponekad prelaze i napon od 1 V i na temelju tih vrijednosti, tj. veoma nepreciznog mjerenja, bilo bi teško utvrditi stanja *LED* dioda, pa postoji velika mogućnost grešaka. Nadalje, desni dio slike 3-2. ističe slučaj kad je *LED* 1 upaljena i vidimo da je pad napona na otporniku 2.56 V. Također, vidimo da, unatoč tome što su ostale *LED* diode isključene, *Arduino* ima pogrešna očitanja napona.

S druge strane, na lijevoj strani slike 3-3. prikazano je mjerenje vrijednosti napona kad su sve *LED* diode isključene i dodani su otpornici, dok je na desnoj strani slike 3-3. slučaj kad je *LED* 1 upaljena, a ostale *LED* diode isključene. Vidimo kako je sad mjerenje puno preciznije.

Mjerenje napona s otpornicima	
Sve LED diode su isključene	Uključena je dioda LED1, a ostale su isključene
0.00,0.00,razlika1 je: 0.00	4.39,1.83,razlika1 je: 2.57
0.00,0.00,razlika2 je: 0.00	0.00,0.00,razlika2 je: 0.00
0.00,0.00,razlika3 je: 0.00	0.00,0.00,razlika3 je: 0.00
**********	**********
0.00,0.00,razlika1 je: 0.00	4.39,1.83,razlika1 je: 2.57
0.00,0.00,razlika2 je: 0.00	0.00,0.00,razlika2 je: 0.00
0.00,0.00,razlika3 je: 0.00	0.00,0.00,razlika3 je: 0.00
**********	**********
0.00,0.00,razlika1 je: 0.00	4.39,1.83,razlika1 je: 2.57
0.00,0.00,razlika2 je: 0.00	0.00,0.00,razlika2 je: 0.00
0.00,0.00,razlika3 je: 0.00	0.00,0.00,razlika3 je: 0.00
**********	**********
0.00,0.00,razlika1 je: 0.00	4.39,1.83,razlika1 je: 2.57
0.00,0.00,razlika2 je: 0.00	0.00,0.00,razlika2 je: 0.00
0.00,0.00,razlika3 je: 0.00	0.00,0.00,razlika3 je: 0.00
**********	**********

*Slika 3-3. Mjerenje napona s otpornicima od 100 k*Ω

U idealnom slučaju se može pretpostaviti da, kada je pad napona veći od 0 V, struja teče kroz strujni krug, a kad je 0 V, struja ne teče, no ovdje nije tako zbog dodanih otpornika (100 kΩ). Konkretno, kada uključimo *izlaz1 Arduina* i uzmemo kratkospojnik, teče jako mala struja koja stvara pad napona na otporniku *R1* od nekoliko mV. Isto tako, ni sam *Arduino* nema 100%-tnu točnost mjerenja, (ponekad mjeri 0.01 ili 0.02 V) te će se iz tog razloga, u programu pretpostaviti da za sve vrijednosti napona manje od 0.1 V stanje *LED* diode je isključeno, a za sve vrijednosti veće od 0.1 V stanje *LED* diode je uključeno. Iz testiranja i u praksi pokazalo se kao veoma točna pretpostavka.

4. Kontrola preko Bluetooth bežične tehnologije

U ovom poglavlju objašnjena je mogućnost kontrole i nadzora uređaja korištenjem *Bluetooth* tehnologije, uz razvoj kontrolne aplikacije na mobilnom uređaju za operacijski sustav *Android*. Taj je dio realiziran na dva načina: korištenjem pojednostavljenog razvojnog sustava *App Inventor* te korištenjem standardnog razvojnog sustava *Android Studio*.

4.1. Android operacijski sustav

Google Android je prvi otvoreni operacijski sustav za mobilne uređaje, tj. sveobuhvatna platforma otvorenog izvornog koda projektirana za mobilne uređaje (mobilni telefoni, tableti, netbook računala, *Google TV*). *Android Inc*. su osnovali Andy Rubin, Rich Miner, Nick Sears i Chris White 2003. god., kako bi razvijali programe za pametne mobilne uređaje. Nakon dvije godine rada, *Google* je odlučio kupiti *Android* te počinju spekulacije o ulasku *Google*-a na tržište pametnih telefona. Osnivači i ključni programeri, osnaženi *Google*-ovim programerima, na tržište donose mobilnu platformu, potpuno prilagodljivu zahtjevima korisnika. *Android* je modularan i prilagodljiv pa postoje slučajevi njegovog prenošenja na razne uređaje. Dakako, aplikacije su ono što će krajnji korisnici smatrati korisnim na *Android*-u. Aplikacija je zapravo jedna datoteka aplikacijskog paketa – *APK* datoteka. Da bi se izrađene aplikacije mogle instalirati na mobilni uređaj, potrebno je u postavkama uređaja omogućiti instalaciju aplikacija koje nisu službeno odobrene od strane *Google*-a. Također, postoji i druga mogućnost. Izrađenu aplikaciju možemo staviti na *Google Play*. To je *Google*-ova mrežna trgovina aplikacija, glazbe i ostalih sadržaja. Za stavljanje aplikacija na *Google Play*-u potrebno je posjedovati *Google* korisnički račun, što se plaća 25 američkih dolara.

4.2. Bluetooth bežična tehnologija

Postoje različiti načini na koje se mogu prenositi podaci između uređaja: žice, radiovalovi, itd. *Bluetooth* tehnologija koristi radiovalove za prijenos podataka. *Bluetooth* bežična tehnologija je inicijalno razvijena kao tehnologija za bežično povezivanje mobilnih aparata i njihovih ekstenzija, ali porast broja i funkcionalnosti elektroničkih uređaja, uz stalnu tendenciju njihove međusobne komunikacije dovodi do potrebne komunikacije između različitih elektroničnih uređaja. I upravo ovdje *Bluetooth* tehnologija preuzima veliku ulogu te ističe svoje prednosti, iz razloga što je bežična i automatizirana, tako da pojednostavljuje proces povezivanja uređaja.

Jedna od definicija *Bluetootha* opisuje ga kao način bežične razmjene podataka između dva ili više uređaja, dok druga definicija objašnjava kako je to način spajanja na osobne bežične mreže [2]. *Bluetooth* tehnologija prijenosa koristi se za daljinsko upravljanje raznim napravama. Da bi bila uključena u *Bluetooth* mrežu, određena naprava treba imati *Bluetooth* adapter (mrežnu karticu), koji sadrži odašiljač i prijamnik.

Frekvencija putem koje se odvija bežična komunikacija je 2,45 GHz, a da bi se izbjegle smetnje koje bi mogle nastati, *Bluetooth* protokol dijeli pojas frekvencija na 79 kanala i mijenja iste 1600 puta u sekundi. *Bluetooth* frekvencije nalaze se u *ISM* (eng. *Industrial Science Medical*) pojasu, koji se obično proteže od 2400 MHz do 2483.5 MHz (tj. 2,4000-2,4835 GHz). *Bluetooth* kanali su raspoređeni po 1 MHz, počevši od 2402 MHz i završava na 2480 MHz [3]. Mogućnost komunikacije između uređaja seže do najveće udaljenosti od oko 100 metara. Budući da se korištenje *ISM* pojasa ne plaća, jasno je da je korisnika mnogo.

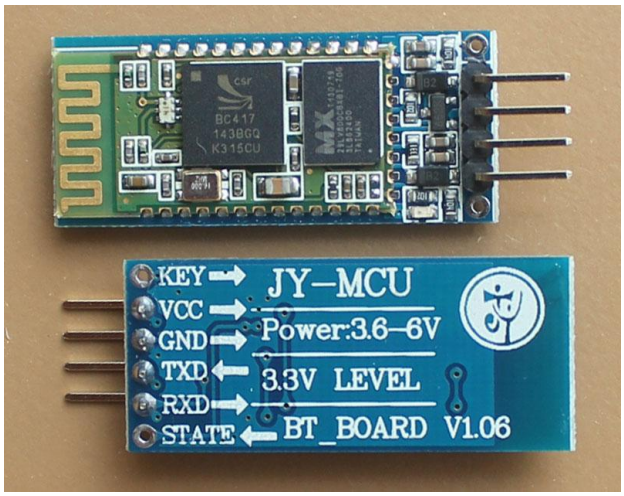
4.2.1. Bluetooth modul HC-06

Ovaj modul omogućuje serijsku komunikaciju između *Arduina* i drugog *Bluetooth* uređaja (u našem slučaju mobilnog uređaja) putem bežične *Bluetooth* veze. *HC-06* modul je modul *SPP* (eng. *Serial Port Profile*) profila, tj. komunikacija između njega i uređaja sa kojima je spojen, temelji se na serijskom prijenosu podataka. Odlika serijske komunikacije je jednostavnost realizacije. Bitovi se kod serijske komunikacije prenose jedan za drugim. Bitovi predstavljaju podatke, a bitovi se prikazuju *TTL* (eng. *Transistor Transistor Logic*) metodom, odnosno naponskim razinama. Na temelju toga, različita kombinacija bitova predstavlja određenu informaciju. Da bi se ostvarila komunikacija, koriste se dva kanala, odnosno vodiča (*Tx i Rx*) kojima se omogućuje dvosmjerna (eng. *full duplex*) komunikacija. Konkretno, pin koji šalje podatke (*Tx*) sa *Arduina*, priključuje se na *Rx* pin *Bluetooth HC-06* modula tako da modul može primati podatke koje *Arduino* šalje. Sukladno tome, na *Rx* pin *Arduina* priključuje se *Tx* pin *Bluetooth* modula kako bi *Arduino* mogao primati podatke sa modula. Fizička (eng. *Media Acces Control*) adresa *HC-06* modula je 20:15:06:24:37:88. Zapisana je u heksadekadskom obliku, sastoji se od 48 bitova, gdje se svakih osam bitova odvaja dvotočkom. Nadalje, spomenuti modul podržava *Bluetooth* verziju 2.0. Svaka verzija ima određene karakteristike. Verzija 2.0 predstavljena je 2004. godine. To je verzija koja je prva predstavila tzv. *EDR* (eng. *Enhanced Data Rate*) tehnologiju. Navedena tehnologija omogućava prijenos podataka brzinom do max. 3 *Mbps*. Prosječna brzina prijenosa je oko 2 *Mbps*. Verzija *Bluetootha* na kojem se testira rad aplikacija, odnosno mobilnog uređaja, je 4.0 (max. brzina do 24 Mbit/s).

Iako je ova verzija novija, kompatibilna je sa verzijom *Bluetooth* modula. Glavna značajka *Bluetooth* 4.0 verzije je mala potrošnja energije (eng. *low energy*) .

Napon napajanja modula može biti u rasponu 3.3 – 6.6 V. Napon pinova za slanje i primanje podataka (*Rx i Tx*) iznosi 3.3 V. Modul je klase 2, a određeni broj klase predstavlja domet unutar kojega modul može emitirati i primati signale. Klasa 2 je definirana za domete do 10-ak metara.

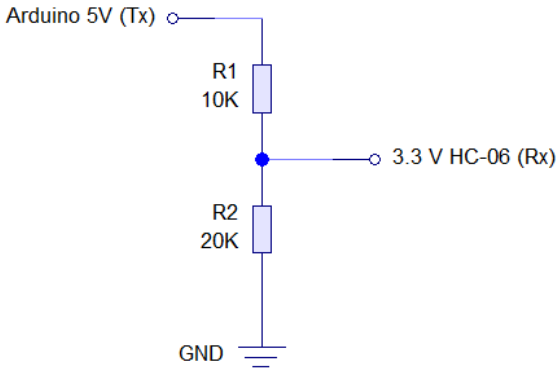
Svi podaci koje *Arduino* šalje, preko *HC-06* modula, vidljivi su mobilnoj aplikaciji te, isto tako, svi podaci koje šaljemo sa mobilne aplikacije preko njega, vidljivi su *Arduinu*. *Arduino* i mobilna aplikacija nemaju nikakvu direktnu vezu, ovdje je *Bluetooth* modul posrednik koji prenosi podatke između njih. Na modulu se nalazi signalna *LED* dioda koja indicira status povezanosti. Kad konstantno svijetli, znači da su uređaji povezani.



Slika 4-1. HC-06 Bluetooth modul

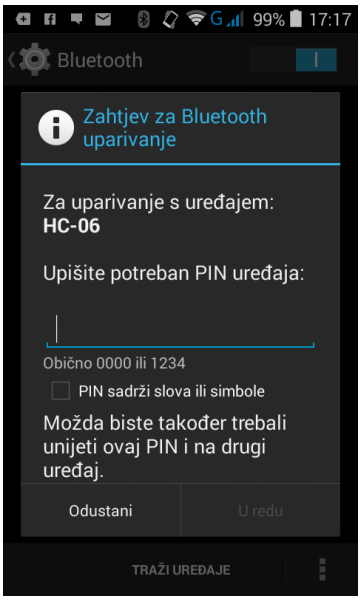
Na slici 4-1. su vidljivi pinovi *HC-06* modula. Pin *VCC* i *GND* su pinovi preko kojih se modulu dovodi napajanje. Potrebno je uzeti u obzir već spomenuti napon pinova za komunikaciju (*TXD* i *RXD*). Naime, potrebno je zaštiti *RXD* pin jer on dobiva podatke sa *Arduina*, naponske razine 5 V. To može oštetiti pin pa je potrebno na neki način smanjiti napon, što je riješeno naponskim djeliteljem. Slika 4-2. prikazuje shemu naponskog djelitelja. Formula za naponski djelitelj iznosi: $U_{izl}=U_{ul} * R2/(R1+R2)$. Konkretno, ovaj djelitelj smanjuje ulazni napon *Uul* (5 V), na željeni izlazni napon *Uizl* (3.3 V). Potrebno je izračunati koliki otpornici se moraju staviti i u kojem omjeru moraju biti, kako bi se dobio željeni izlazni napon. Vrijednost jednog otpora proizvoljno biramo (10 kΩ), a vrijednost drugog računamo iz formule: $R2=U_{izl}*R1/(U_{ul}-U_{izl})$. Vrijednost drugog otpornika ispada 20 kΩ. Ovo su proizvoljne

vrijednosti otpornika i tu se, naravno, može uzeti bilo koja kombinacija otpornika koja daje izlazni napon od 3.3 V.



Slika 4-2. Naponski djelitelj

Bluetooth pruža brojne mogućnosti, no bez obzira što se želi učiniti, prvi korak je uparivanje dvaju uređaja koji žele razmjenjivati podatke koristeći *Bluetooth* komunikaciju. Pritom je nužno provjeriti je li dodatak ili uređaj koji se želi upariti uključen i fizički dostupan, odnosno u području dometa. Uparivanje se pokreće ručno, od strane korisnika uređaja. Da bi došlo do uparivanja, mora se razmijeniti lozinka između dva uređaja, čime se potvrđuje da oba korisnika pristaju na uparivanje. Uređaji se mogu upariti u postavkama mobilnog uređaja. Za uparivanje sa *HC-06* modulom potrebno je unijeti lozinku za identifikaciju (slika 4-3.).

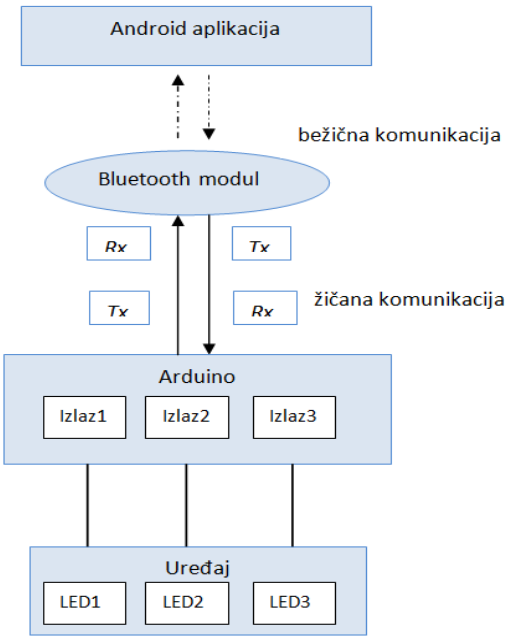


Slika 4-3. Uparivanje sa HC-06 modulom

U *Bluetooth* terminologiji veza se stvara između nadređenog, gospodara (eng. *Master*) i rob (eng. *Slave*) uređaja [3]. Gospodar je uređaj koji inicijalizira komunikaciju, u našem slučaju *Android* aplikacija, odnosno mobilni uređaj je nadređeni uređaj, a *Bluetooth* modul *HC-06* je rob.

4.3. Protokol komunikacije

Bluetooth modul je komunikacijski posrednik između aplikacija i *Arduina*. Ta veza je prikazana na slici 4-4. Naime, jednom kad se uspostavi *Bluetooth* veza između aplikacije i *Bluetooth* modula, putem njega *Arduino* i aplikacija mogu komunicirati. *Android* aplikacijom šaljemo određene podatke, *Arduino* ih čita te na temelju njih upravlja svojim izlazima, odnosno *LED* diodama. Podaci na temelju kojih *Arduino* upravlja izlazima su znakovi *a*, *b* (za *izlaz1*), *c*, *d* (*izlaz2*) i *e*, *f* (*izlaz3*). Kad *Arduino* pročita znak *a*, uključuje *izlaz1* i drži ga uključenog. Kad dobije znak *b*, isključi *izlaz1* i drži ga isključenog. Za upravljanje *izlazom2* koristimo znak *c* (za uključivanje) i znak *d* (za isključivanje). Za *izlaz3* koristimo znak *e* za uključiti, a znak *f* za isključiti. Na koji način ove podatke šaljemo preko aplikacija, bit će objašnjeno pod opisom programskog dijela aplikacija za svaku aplikaciju posebno. Isto tako, *Arduino* čita stanja izlaza i stanja *LED* dioda, generira određene podatke i šalje ih aplikaciji, a na temelju njih aplikacija prikazuje stanja. Na koji način to radi i koji su to podaci koje *Arduino* šalje, objašnjeno je pod opisom *Arduino* programa za aplikacije (poglavlje 4.4). Bitno je napomenuti da *Arduino* stanja izlaza i *LED* dioda konstantno čita u petlji (*loop*) i konstantno ih šalje aplikaciji, a u aplikaciji se po potrebi šalju podaci *Arduinu* kojim možemo kontrolirati izlaze.



Slika 4-4. Protokol komunikacije

4.4. Opis Arduino programa za aplikacije

Program na *Arduinu* prilagođen je za rad sa aplikacijom u *App Inventoru* i *Android Studiu*. U daljnjem tekstu objašnjeni su pojedini dijelovi programskog koda na *Arduinu*.

U početku programa deklariramo varijablu *dolazno* tipa *char*, u koju se spremaju dolazni podaci (znak), koji se šalju sa aplikacije, a *Arduino* ih čita preko *HC-06 Bluetooth* modula. Deklariramo i inicijaliziramo varijable *izlaz1, izlaz2 i izlaz3* kao cjelobrojne vrijednosti i dodjeljujemo im vrijednosti redom 5, 6, 7. Deklariramo varijable *razlika1, razlika2 i razlika3* koje predstavljaju razliku napona na otpornicima *R1, R2 i R3*. Zbog preciznijeg mjerenja uzimamo decimalne brojeve, odnosno *float* tip podataka. Deklariramo i inicijaliziramo varijablu *pretvorba,* kojom množimo pad napona na otpornicima kako bi umjesto vrijednosti (0-1023) dobili veličine preračunate u volte (0-5 V).

Početni dio programa

```
//za spremanje dolaznih podataka(upravljački podaci-a,b,c,d,e,f)
char dolazno;

//pinovi za izlaz1,izlaz2,izlaz3
int izlaz1  = 5;
int izlaz2  = 6;
int izlaz3  = 7;

//za spremanje pada napona na R1,R2,R3
float razlika1;
float razlika2;
float razlika3;

//koristimo kod pretvaranja analognih vrijednosti (0-1023) u volte (0-5 V)
float pretvorba = (5.0 / 1023.0);
```

U *setup* dijelu programa pomoću funkcije *Serial.begin* započinje se serijska komunikacija između modula *HC-06* i *Arduina* i postavlja se brzina prijenosa podataka od 9600 bitova po sekundi. Kod korištenja funkcije *Serial*, pinovi *Rx* (0) i *Tx* (1) služe za komunikaciju i ne mogu se koristiti u druge svrhe.

Pomoću funkcije *pinMode* konfiguriraju se pinovi, odnosno postavljaju se kao ulazni ili izlazni. Sintaksa za ovu funkciju je: *pinMode (pin, mode)*. Pod *pin* se stavlja broj pina kojeg želimo konfigurirati, a pod *mode*, definiramo da li je pin ulazni (*INPUT*) ili izlazni (*OUTPUT*). U ovom slučaju, digitalne pinove 5, 6 i 7 postavili smo kao izlazne.

Iako koristimo svih šest analognih pinova (*A0-A5*) kao ulazne, nije ih potrebno deklarirati, jer se bez deklaracije pinovi automatski smatraju kao ulazni.

Funkcija *setup*

```
void setup()
{
     //započinjemo serijsku komunikaciju
     Serial.begin(9600);
     //postavljamo pinove 5,6,7 kao izlazne
     pinMode(izlaz1, OUTPUT);
     pinMode(izlaz2, OUTPUT);
     pinMode(izlaz3, OUTPUT);
}
```

U ovom dijelu, tj. *loop* funkciji najprije se poziva funkcija *citajstanje*, u kojoj *Arduino* čita stanja i*zlaza1, 2 i 3* i mjeri napon na temelju kojeg očitava stanja *LED* dioda. Funkcija *citajstanje* je objašnjena niže u tekstu. Funkcija *Serial.available* vraća vrijednost 1, ukoliko postoje podaci za čitanje sa serijskog porta. Ako ti podaci postoje, oni se pomoću funkcije *Serial.read* čitaju i spremaju u varijablu *dolazno* koja je deklarirana na početku.

Pomoću *switch-case* uvjetnog grananja, uspoređuju se vrijednosti varijable *dolazno* sa slučajevima koji su zadani pod *case.* U slučaju da je vrijednost varijable *dolazno* znak *a* , *Arduino* uključi *izlaz1* i drži ga uključenog tako dugo dok ne dobije znak *b.* Na znak *b* isključuje *izlaz1* i drži ga isključenog. *Izlazima 2 i 3 Arduina* upravlja se na isti način po principu koji je već ranije opisan (*izlaz2 - c, d, izlaz3 - e, f*). Za upravljanje pinovima koristimo funkciju *digitalWrite,* a parametri koje prosljeđujemo su broj pina i vrijednost napona (*HIGH*- 5 V ili *LOW*- 0 V). Pomoću *delay* vremenske funkcije zaustavljamo izvođenje programa na određeno vrijeme (u milisekundama).

Funkcija *loop*

```
void loop()
{
   //pozivamo funkciju u kojoj se čitaju stanja izlaza i LED dioda
   citajstanje();

       //ako ima podataka na serijskom portu,
       //čitaj ih i spremaj u varijablu dolazno
       if (Serial.available() > 0)  {
         dolazno = Serial.read();
       }
         // uspoređuj varijablu dolazno sa slučajevima pod "case"
        switch (dolazno)
        {
              //ako je vrijednost a, uključi izlaz1
             case 'a': digitalWrite(izlaz1, HIGH);
               break;
             //ako je vrijednost b, isključi izlaz1
             case 'b': digitalWrite(izlaz1, LOW);
               break;
             //ako je vrijednost c, uključi izlaz2
             case 'c': digitalWrite(izlaz2, HIGH);
               break;
              //ako je vrijednost d, isključi izlaz2
             case 'd': digitalWrite(izlaz2, LOW);
               break;
              //ako je vrijednost e, uključi izlaz3
             case 'e': digitalWrite(izlaz3, HIGH);
               break;
              //ako je vrijednost f, isključi izlaz3
             case 'f': digitalWrite(izlaz3, LOW);
               break;
        }
   delay (100);
}
```

U funkciji *citajstanje* najprije čitamo stanje digitalnih izlaza *Arduina* pomoću funkcije *digitalRead*. Ukoliko je na pinu 5 V, pomoću funkcije *Serial.print* ispisujemo *prvion*, u slučaju da je na pinu 0 V, ispisujemo *prviof*. Te podatke koji se ispisuju možemo vidjeti u *Serial monitoru*, a isto tako vidi ih i *Bluetotoh* modul. Također, te podatke preko *Bluetooth* modula aplikacija prima i na temelju njih prikazuje stanja izlaza *Arduina*. Za čitanje stanja ostalih izlaza, koristimo isti princip, samo što se ispisuju druge vrijednosti kako bi aplikacija znala o kojem pinu se radi (*izlaz2- drugion, drugiof, izlaz3- trecion, treciof*).

Nakon čitanja stanja izlaza, čitamo stanja *LED* dioda. Funkcija *analogRead(A0)* čita vrijednost analognog ulaza (napon) na pinu *A0* i tu vrijednost spremamo u varijablu *veci1*. Ime varijable označuje da se ovdje, kad teče struja, nalazi veći napon. Paralelno tome, u varijablu *manji1* sprema se vrijednost napona na analognom pinu *A1*. Kako bi dobili pad napona na otporniku *R1*, potrebno je oduzeti napon na pinu *A1* i pinu *A0*. Tu razliku, odnosno njezinu

vrijednost spremamo u varijablu *razlika1* i množimo sa varijablom *pretvorba* da dobijemo vrijednost izraženu u voltima. Na temelju *razlike1* pretpostavljamo stanje *LED 1* (općenito, za napon veći od 0,1 V *LED* dioda svijetli, u suprotnom ne). Znači, kad je *LED 1* uključena, *Arduino* ispisuje, odnosno šalje podatak oblika *LED1on* čime aplikacija signalizira da je *LED 1* uključena. Kad je isključena, šalje *LED1of* i aplikacija prikazuje isključeno stanje *LED 1*. Isti princip koristi se i za ostale *LED* diode samo *Arduino* ispisuje različite vrijednosti.

Funkcija *citajstanje*

```
//funkcija u kojoj se čitaju stanja (izlaza i LED dioda) i ispisuju,
//odnosno šalju na serijski port
voidcitajstanje()
{
     Serial.print("*"); //označava početak podataka

     //čitanje stanja izlaza1, za ispis-->Serial.print
     if (digitalRead(izlaz1)) Serial.print("prvion"); //uključen
     else                     Serial.print("prviof"); //isključen

     // služi za odvajanje podataka (stanja izlaza i LED dioda)
     Serial.print (":");

     //čitanje stanja za izlaz2
     if (digitalRead(izlaz2)) Serial.print("drugion"); //uključen
     else                     Serial.print("drugiof"); //isključen
     Serial.print(":");//odvajanje podataka

     //čitanje stanja za izlaz3
     if (digitalRead(izlaz3)) Serial.print("trecion");  //uključen
     else                     Serial.print("treciof");  //isključen
     Serial.print (":"); //odvajanje podataka

     //čitanje stanja LED1
     float veci1  = analogRead(A0);
     float manji1 = analogRead(A1);
     //Pad napona na R1
     razlika1 = (veci1 - manji1) * pretvorba;
     //ako je pad napona veći od 0,1V LED-ice svijetle u suprotnom ne
     if (razlika1 > 0.1 ) Serial.print("LED1on"); // uključena
     else                 Serial.print("LED1of"); //isključena
     Serial.print (":"); //odvajanje podataka

     //čitanje stanja LED2
     float veci2  = analogRead(A2);
     float manji2 = analogRead(A3);
     //Pad napona na R2
     razlika2 = (veci2 - manji2) * pretvorba;
     if (razlika2 > 0.1 ) Serial.print("LED2on"); // uključena
     else                 Serial.print("LED2of"); //isključena
     Serial.print (":"); // odvajanje podataka

     //čitanje stanja LED3
     float veci3  =  analogRead(A4);
     float manji3 =  analogRead(A5);
     //Pad napona na R3
     razlika3 = (veci3 - manji3) * pretvorba;
     if (razlika3 > 0.1)  Serial.print("LED3on"); //uključena ("LED3on")
     else                 Serial.print("LED3of"); //isključena

     Serial.print(":");  //odvajanje podataka
     Serial.print("#");  //označava kraj podataka
     Serial.println();   // prebacivanje u novi red
     delay(150);
}
```

```
*prviof:drugiof:treciof:LED1of:LED2of:LED3of:#
*prviof:drugiof:treciof:LED1of:LED2of:LED3of:#
*prviof:drugiof:treciof:LED1of:LED2of:LED3of:#
*prviof:drugiof:treciof:LED1of:LED2of:LED3of:#
```

Slika 4-5. Podaci koje generira funkcija citajstanje

Na slici su prikazani podaci koje *Arduino* ispisuje u funkciji *citajstanje,* odnosno šalje aplikaciji (slika prikazuje primjer kad su svi izlazi i sve *LED* diode isključeni). U aplikaciji se ti podaci dohvaćaju i na temelju njih se prikazuju stanja.

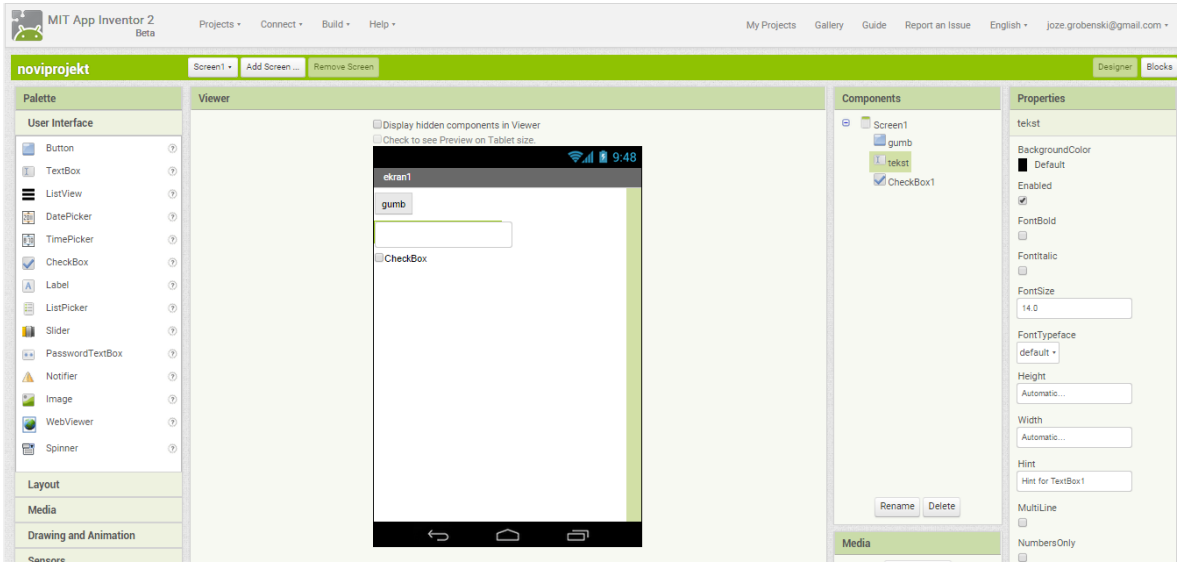
Vidimo kako svaki red počinje znakom *, a završava znakom #. Ovi znakovi prvenstveno su nam potrebni kako bi u aplikaciji izrađenoj u *Android Studiu* identificirali podatke koji nas zanimaju, odnosno njihov početak i kraj, a način na koji se to odvija, objašnjen je u poglavlju 4.6.1.

Podaci između znakova **:** predstavljaju stanja izlaza *Arduina* i stanja *LED* dioda. Oni nam također služe kako bi u aplikaciji izrađenoj u *Android Studiu* odvojili podatke koji nam trebaju. Prva tri podatka odvojena znakom**:** predstavljaju stanja izlaza (*prviof, drugiof, treciof*), a preostala tri stanja *LED* dioda(*LED1of, LED2of, LED3of*).

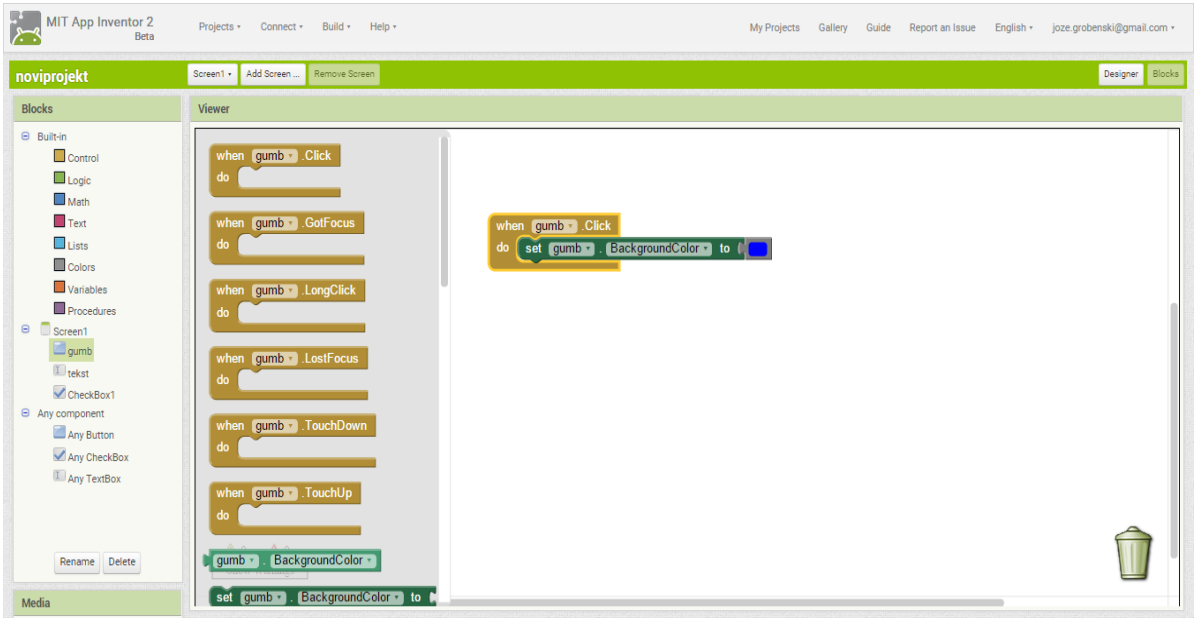
4.5. App Inventor

App Inventor za *Android* operacijski sustav je razvojno okruženje, izvorno razvijena od strane *Google*-a. Međutim, kasnije ga preuzima *Massachusetts Institute of Technology* (*MIT*) te trenutno nosi naziv *MIT App Inventor*. Pobliže, *MIT App Inventor* je alat za izradu aplikacija, namijenjen svima i za njegovo korištenje nije potrebno poznavanje programskog jezika. Sastoji se od dva dijela, *designer* i *block* dijela, koji su vidljivi na slikama 4-6. i 4-7. U *designer* djelu su specificirane komponente programa, odnosno u tom djelu se kreira vizualni dio aplikacije [4]. Komponente mogu biti vidljive (gumbi, slike i sl.) te se nalaze na simuliranom zaslonu te nevidljive (*Bluetooth* klijent, obavijest- *notifier* sl.). U *block* dijelu se elementima, koji su prethodno definirani u vizualnom dijelu, dodaju određene funkcije. Ovdje se stvara programska logika. Blokovi određuju kako se komponente trebaju ponašati, a slažu se poput slagalice.

App Inventor nije potrebno instalirati na računalo iz razloga što se nalazi na web poslužitelju. Dovoljno je samo da se korisnik prijavi putem *Google* računa te koristi *App Inventor* preko web preglednika.



Slika 4-6. App Inventor designer dio



Slika 4-7. App Inventor block dio

App Inventor nam pruža mogućnost istovremenog testiranja aplikacije kako je izgrađujemo. Ukoliko korisnik nema mobilni uređaj, aplikaciju i njezinu funkcionalnost je moguće pratiti pomoću *Android* emulatora, tj. softvera na računalu, koji se ponaša identično kao i mobilni uređaj.

Jednom razvijena aplikacija se može dijeliti s drugim korisnicima. Naime, moguće je dijeljenje u izvršnom obliku (.apk), odnosno spremi se na računalo s navedenom ekstenzijom te se potom instalira na mobilni uređaj. Isto tako, druga opcija je dijeljenje u obliku izvornog koda (.aia), što omogućuje da korisnik aplikaciju otvori u *App Inventoru* te ima mogućnost mijenjanja aplikacije.

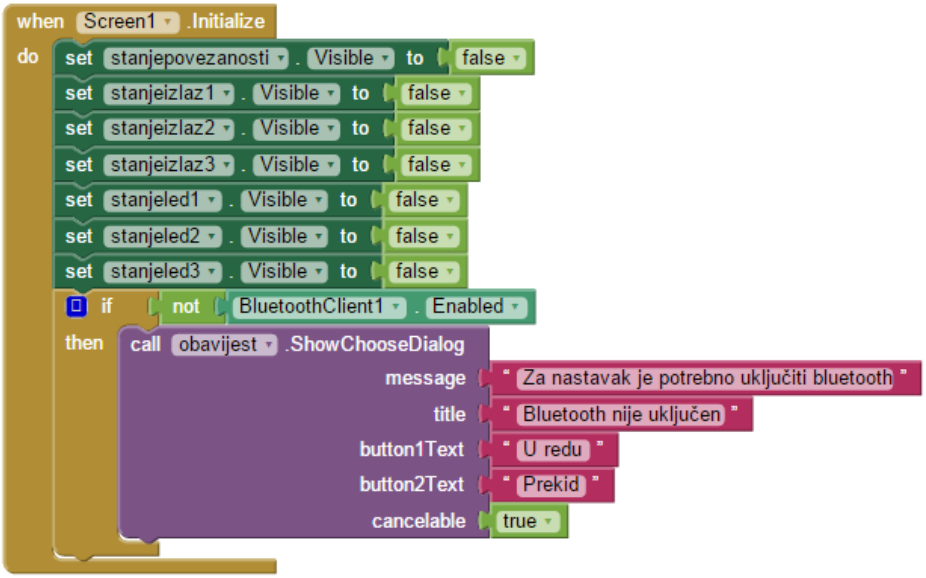
4.5.1. Princip rada aplikacije u App Inventoru

U aplikaciji se prikazuju stanja izlaza *Arduina* (*izlaz1, izlaz2 i izlaz3*) i stanja *LED* dioda. Stanja se prikazuju gumbima. Gumb zelene boje simbolizira da je *izlaz/LED* dioda uključen, dok gumb crvene boje simbolizira da je *izlaz/LED* dioda isključen. Kako bi slali određene naredbe kojima možemo kontrolirati izlaze, također nam služe gumbi. Pritiskom na određeni gumb, šalje se određeni znak koje *Arduino* čita i ovisno o znaku koji pročita upravlja svojim izlazima. Već je prije spomenuto da su to znakovi *a, b, c, d , e* i *f*. Istovremeno, *Arduino* čita stanja izlaza i stanja *LED* dioda i te podatke šalje na aplikaciji. Na temelju tih podataka u

aplikaciji se prikazuju stanja *izlaza i LED* dioda, a na koji način se dohvaćaju i obrađuju u aplikaciji, navedeno je u daljnjem tekstu.

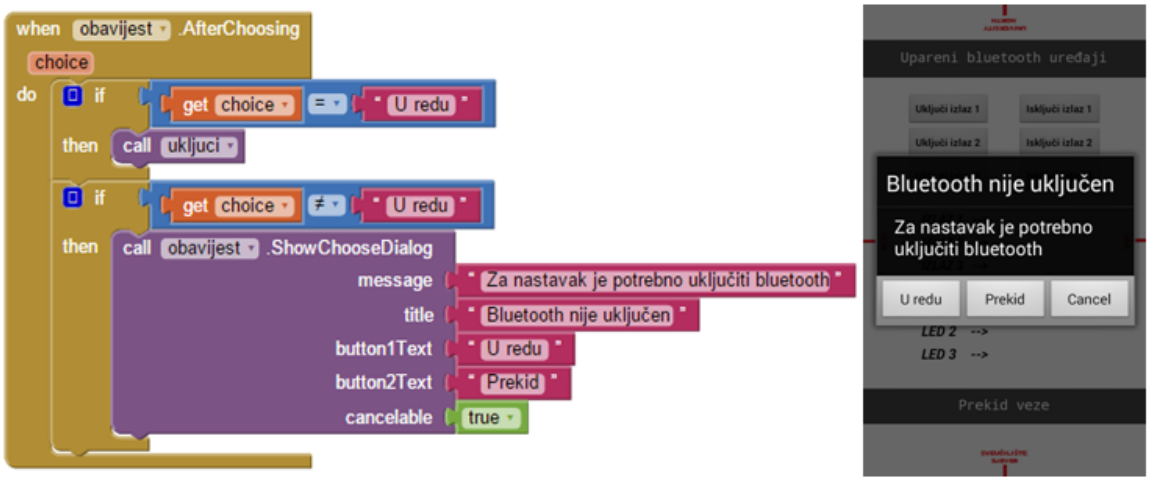
4.5.2. Logički dio aplikacije u App Inventoru

Logički dio aplikacije u *App Inventoru* obuhvaća *block* dio, koji je već prethodno u tekstu spomenut i objašnjen. U nastavku je na nekoliko primjera objašnjen princip rada logičkog dijela. Naime, na slici 4-8. je dio bloka u kojem je definirano da se prilikom pokretanja aplikacije sakriju, odnosno ne prikažu gumbi koji signaliziraju stanja i tekst koji govori status povezanosti sa *HC-06* modulom. Ukoliko *Bluetooth* na mobilnom uređaju nije uključen, na ekranu se pojavljuje obavijest da je za nastavak potrebno uključiti *Bluetooth*.



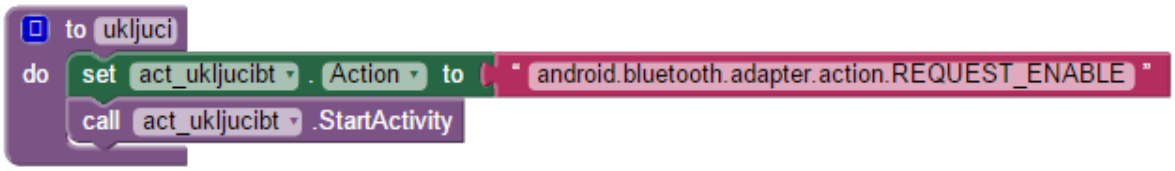
Slika 4-8. Početni zaslon aplikacije

U bloku, prikazanom na slici 4-9., definirano je da se obavijest o uključenju *Bluetooth-a* prikazuje svaki put ukoliko odabir nije *U redu*. Nakon što korisnik odabere *U redu*, poziva se procedura *ukljuci*, u kojoj se započinje nova aktivnost i pomoću naredbe *android.bluetooth.adapter.action.REQUEST_ENABLE* se *Bluetooth* automatski uključuje.



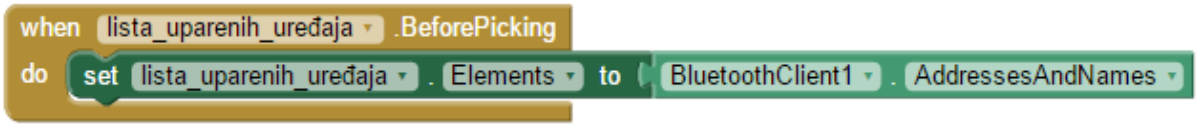
Slika 4-9. Obavijest za uključenje Bluetooth-a

Sljedeći prikaz na slici 4-10. predstavlja proceduru *ukljuci* koja uključuje *Bluetooth*.



Slika 4-10. Procedura koja uključuje Bluetooth

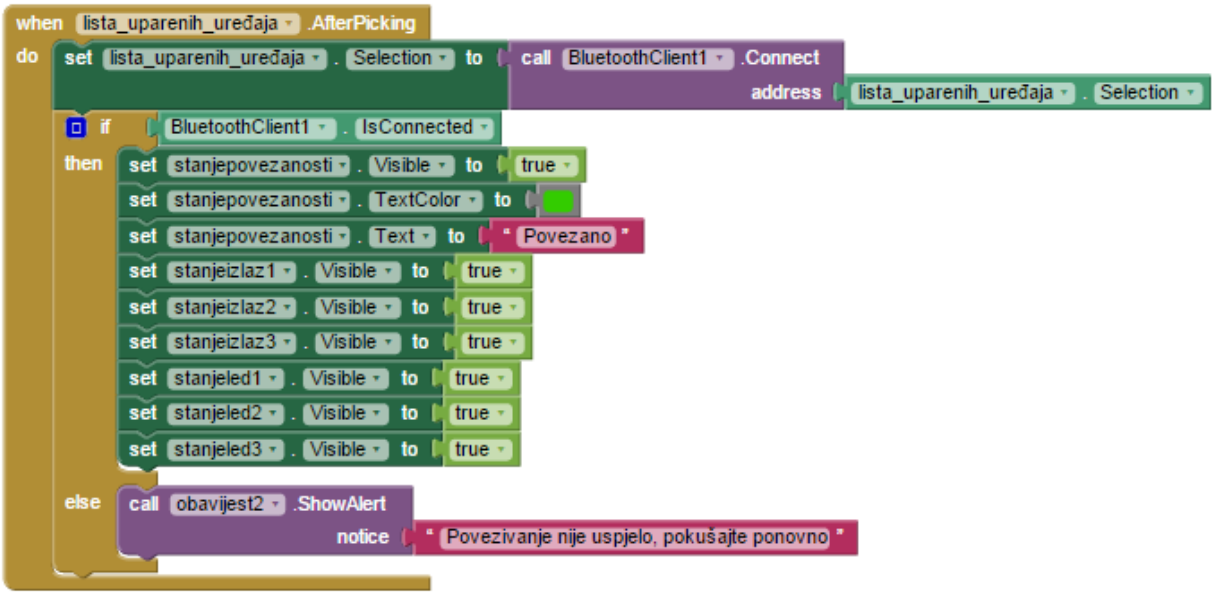
Lista uparenih uređaja predstavlja listu, odnosno popis uparenih *Bluetooth* uređaja. Klikom na nju, elemente liste postavlja na pronađene uparene *Bluetooth* uređaje u okolini i prikazuje informacije o njima (*MAC* adresa i ime uređaja). Taj primjer prikazan je na slici 4-11.



Slika 4-11. Lista uparenih Bluetooth uređaja

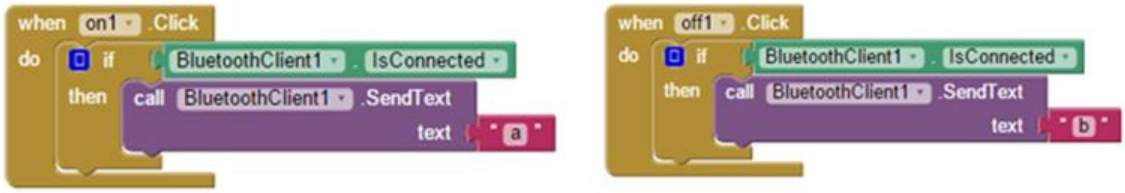
Nakon što lista prikaže uparene *Bluetooth* uređaje, može se definirati što će se dogoditi kada se odabere, odnosno klikne na element liste (*Bluetooth* uređaj). Nakon odabira elementa sa liste, *Bluetooth* klijent započinje proces povezivanja sa *Bluetooth* uređajem kojeg smo odabrali na listi. Ukoliko je povezivanje uspješno, na ekranu se prikazuje informacija o stanje povezanosti

te gumbi koji signaliziraju stanja izlaza *Arduina* i stanja *LED* dioda. Ukoliko povezivanje nije uspjelo, na ekran se ispisuje poruka *Povezivanje nije uspjelo, pokušajte ponovno* (slika 4-12.).



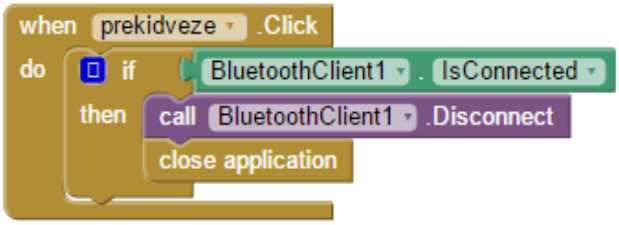
Slika 4-12. Uspostavljanje Bluetooth veze

Nadalje, bitno je spomenuti na koji način šaljemo kontrolne podatke (*a, b, c, d, e, f*) na temelju kojih *Arduino* upravlja svojim izlazima. Na slici 4-13. prikazani su gumbi imena *on1* i *off1*. Njima šaljemo znak *a* ili znak *b*. Na isti način pomoću gumba šaljemo ostale podatke (*c, d, e, f*).



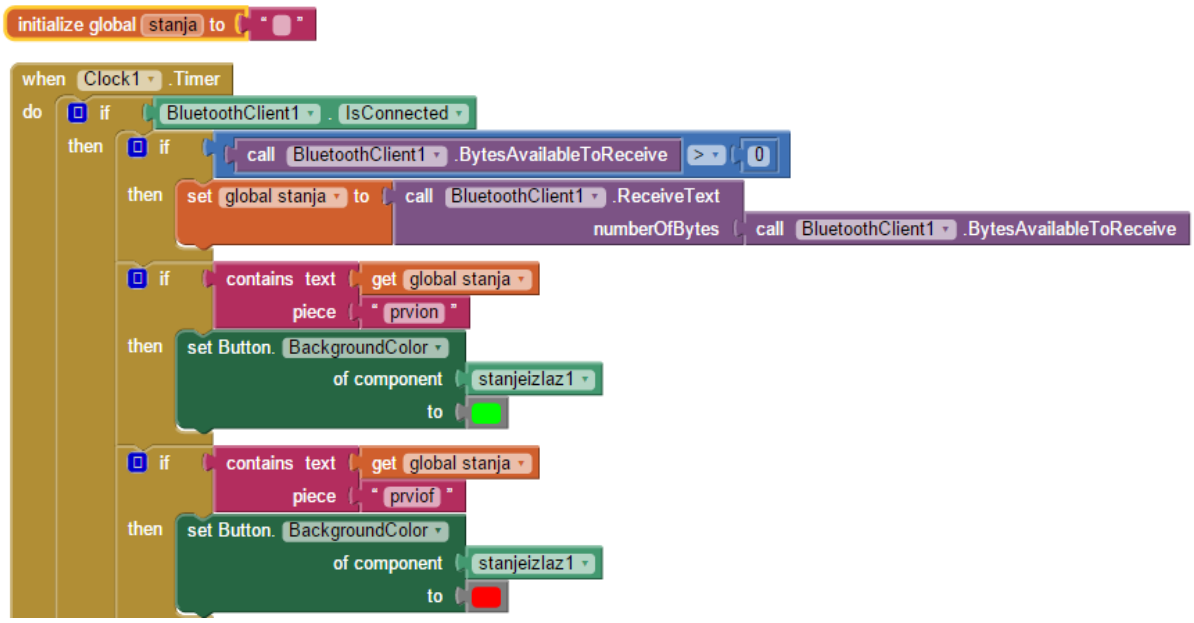
Slika 4-13. Gumbi kojima šaljemo podatke Arduinu

Ukoliko kliknemo na gumb *prekidveze*, a *Bluetooth* je trenutno povezan, on prekida vezu i izlazi iz aplikacije. Kad *Bluetooth* nije povezan, gumb nema nikakvu funkciju (slika 4-14).



Slika 4-14. Gumb kojim prekidamo vezu

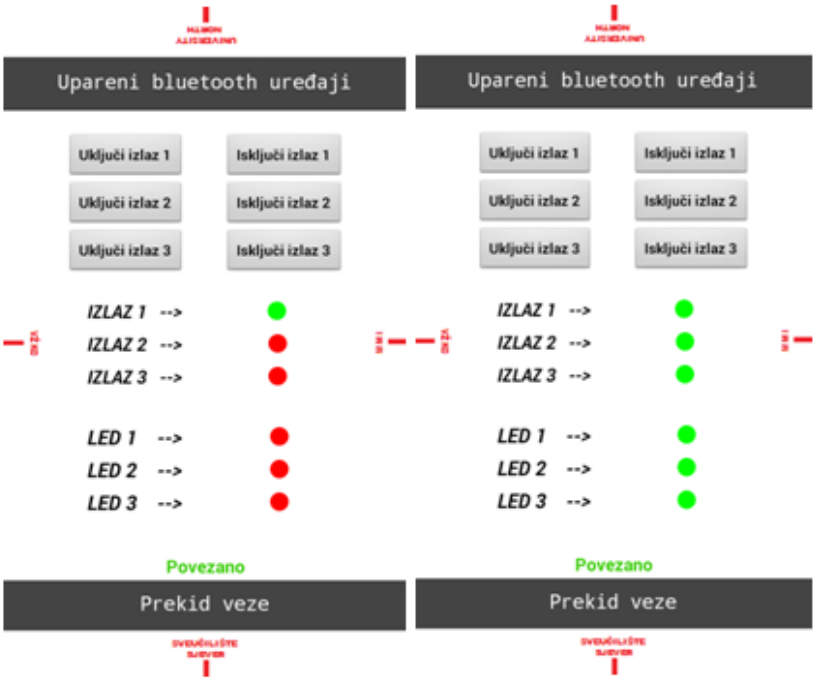
Dio bloka prikazan na slici 4-15 služi za dohvat podataka sa *Bluetooth* modula. Najprije je inicijalizirana globalna varijabla imena *stanja*. U ovu varijablu spremamo podatke koje nam šalje *Arduino*, odnosno generira funkcija *citajstanje*. Varijabla zapravo dobiva tekst, odnosno niz znakova pa se može reći da je tipa *string*. *Timer* služi za periodičko provjeravanje ima li podataka sa *Bluetooth* modula za čitanje (svakih 300 ms). Već je u prethodnom tekstu spomenuto da dolazni podaci koje aplikacija prima predstavljaju stanja izlaza i *LED* dioda. Ovisno o dolaznim podacima gumbi bojama signaliziraju ta stanja.



Slika 4-15. Primanje podataka i prikaz stanja

U *App Inventor* aplikaciji nisu nam potrebni posebni znakovi (*, #, **:**) kojim odvajamo stanja, već aplikacija izvlači samo one podatke koji su joj potrebni dok znakove (*, #, :) zanemaruje. Iz primjera sa slike izdvojen je dio koji procesira podatke vezane za *izlaz1*. Ukoliko dolazni podaci sadrže tekst *prvion*, gumb koji signalizira stanje *izlaza1* mijenja boju u zeleno. Ukoliko dolazni podaci sadrže tekst *prviof*, gumb mijenja boju u crveno.

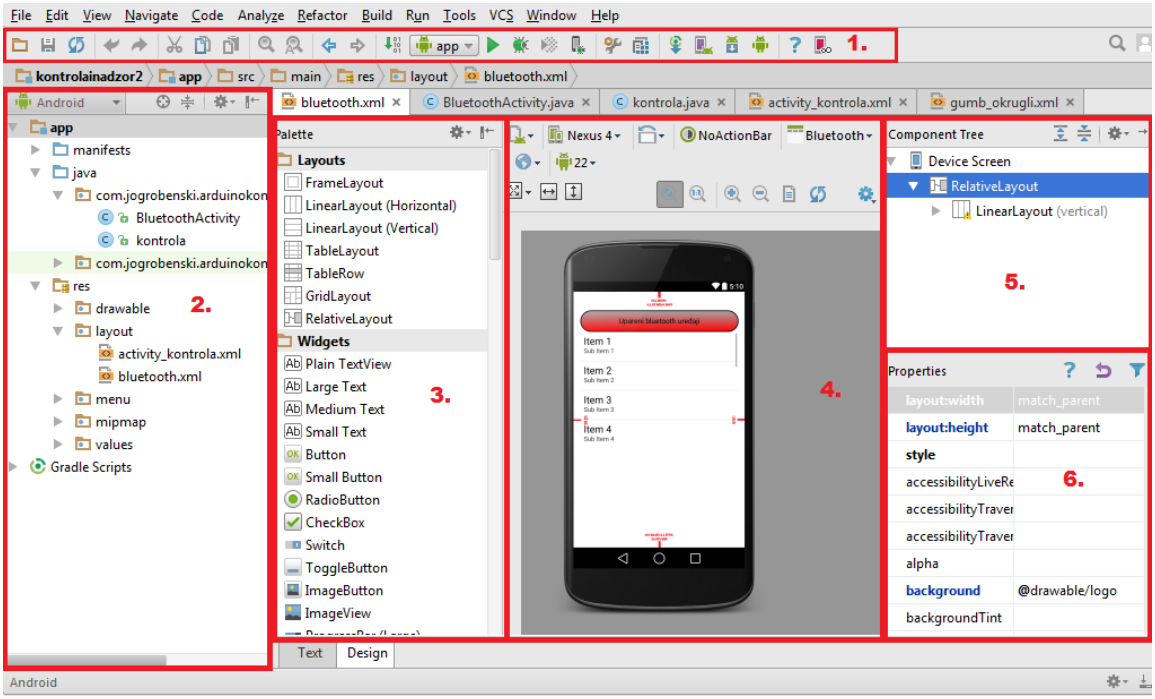
Sljedeća slika 4-16. prikazuje izgled aplikacije u *App Inventoru*. Lijevo na slici je prikazana simulacija „pregorjele“ *LED* 1 diode. S druge strane, desno na slici su prikazane *LED* diode uključene programski, preko aplikacije.



Slika 4-16. Izgled aplikacije u App Inventoru

4.6. Android Studio

Android Studio pruža sve potrebno kako bi se mogla razviti *Android* aplikacija. Uključuje *IDE* razvojno okruženje i *Android SDK* (eng. *Software Development Kit*) alate. Zamijenio je *Googlovo* prethodno primarno okruženje za *Android* aplikacije, *Eclipse*. Nudi niz mogućnosti za fleksibilan sustav, izgradnju višestruke *APK* datoteke, predloške koda, alate za učinkovitost, iskoristivost, kompatibilnost i sl. Na slici 4-17. istaknuti su dijelovi koji čine vizualni dio *Android Studio* okruženja. Pod brojem jedan je alatna traka, zatim sadržaj projekta *Android* aplikacije (raspored datoteka). Broj tri prikazuje paletu koja sadrži razne elemente korisničkog sučelja (gumb, liste i sl.), a pod brojem četiri je grafičko sučelje u kojem se vidi izgled aplikacije. Nakon toga je popis elemenata na zaslonu te su pod brojem šest prikazana svojstva elemenata (gumbi, liste, tekst) i mogućnost mijenjanja njihovih svojstava (boja, veličina, margine i sl.).



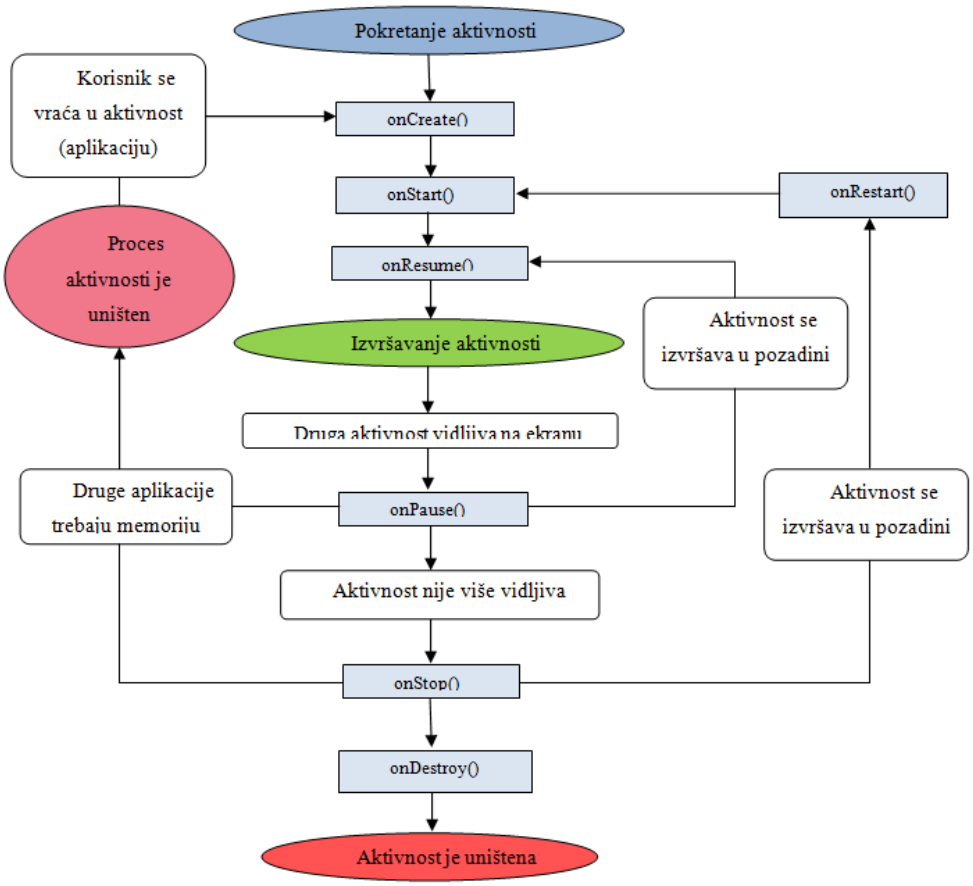
Slika 4-17. Prikaz dijelova Android Studia

U programiranju, programsko sučelje za aplikacije (*API*, eng. *Application Programming Interface*) je skup rutina, protokola i alata za izgradnju aplikacija. Definira funkcionalnosti aplikacije, tj. dobar *API* olakšava izradu programa pružanjem svih blokova, koji se zatim stavljaju zajedno u cjelinu. Određena razina *API-ja* je za određenu verziju *Androida*, pa je tako *API* razina (eng. *level)* 19 predviđena kao najbolja za *Android* verziju *KitKat*, odnosno *Android*

4.4. Kod izrade projekta, tj. aplikacije, moramo birati minimalnu razinu *API-ja*, što znači da aplikaciju podržavaju sve *Android* verzije od 2.3. pa nadalje, iz razloga što je minimalan izabrani *API* 10, koji je predviđen upravo za 2.3. verziju *Android-a* (verzija imena *Gingerbread*).

4.6.1. Opis programskog koda izrađene aplikacije u Android Studiu

Aplikacija se sastoji od dvije aktivnosti (eng. *Activity*). Aktivnost možemo promatrati kao zaseban zaslon, odnosno ekran aplikacije. Prva aktivnost naziva se *BluetoothActivity*, a druga *kontrola*. Da bi bilo jednostavnije shvatiti programski dio aplikacije, potrebno je opisati životni ciklus aktivnosti. Vidimo na slici 4-18. da kod pokretanja aktivnosti od početka (*Pokretanje aktivnosti*), aktivnost prolazi kroz tri ciklusa, odnosno metode (*onCreate, onStart* i *onResume*). Tek kad aktivnost prođe kroz ta tri ciklusa, ona je korisniku vidljiva na zaslonu što na slici prikazuje polje *Izvršavanje aktivnosti*. Kad se pojavi neka druga aktivnost (npr. dolazni poziv) koja ima veći prioritet, aktivnost prelazi u stanje *onPause*. Kad aktivnost većeg prioriteta prestane, aktivnost, prije nego što je ponovno vidljiva na zaslonu mobilnog uređaja, prolazi kroz stanje *onResume*. Kad zatvorimo aktivnost, ona više nije vidljiva na ekranu, ona se nalazi u stanju *onStop*. Nadalje, u ciklusu *onStop*, ukoliko je potrebno osloboditi memoriju za neke važnije procese, sustav može uništiti aktivnost. Za njezino ponovno pokretanje, aktivnost prolazi kroz početne tri faze prije nego što je vidljiva korisniku.



Slika 4-18. Životni ciklus aktivnosti

Svaki zaslon (aktivnost) ima vizualni (grafički) dio i programski dio. Tako se, primjerice, za aktivnost *kontrola,* programski kod koji definira izgled aktivnosti nalazi u datoteci *kontrola.xml*, a programski dio u datoteci *kontrola.java*. Programski dio omogućuje interakciju elemenata definiranih u grafičkom sučelju, sa stvarnim svijetom (može se definirati što se događa kod klika na gumb i slično). U *onCreate* metodi povezuje se vizualni dio sučelja (eng. *layout*) sa programskim (logičkim) dijelom. Grafički izgled sučelja napisan je u datoteci sa ekstenzijom .xml.

XML (eng. *Xtensible Markup Language*) je skup pravila za kodiranje dokumenata u strojno čitljivom obliku. Konkretno, koristi se i u *Android Studiu*, za definiranje vizualnog dijela, odnosno za opis vizualnih elemenata (gumb, lista i sl). Na primjer, prilikom povlačenja gumba na ekran, uz taj gumb se prikaže i XML zapis koji ima nekoliko informacija o njemu (visina, identifikator i sl.). Osim grafičkog (layout) dijela, XML se koristi i u *manifest* datoteci. Datoteka *manifesta* povezuje sve komponente. To je datoteka koja objašnjava od čega se aplikacija sastoji, koji su njeni glavni gradivni blokovi, tj. u njoj se definiraju koja dopuštenja aplikacija zahtijeva i slično [5]. *Manifest* datoteka izrađene aplikacije nalazi se u prilogu rada (prilog 2).

```
<Button
    android:layout_width="110dp"
    android:layout_height="33dp"
    android:text="Isključi  izlaz 1"
    android:id="@+id/iz1off"
    android:background="@drawable/gumb_crveni"
    android:width ="110dp"
    android:layout_marginLeft="5dp"
    android:layout_weight="0.12"
    android:textSize="12dp"  />
```

Slika 4-19. Primjer XML koda za gumb

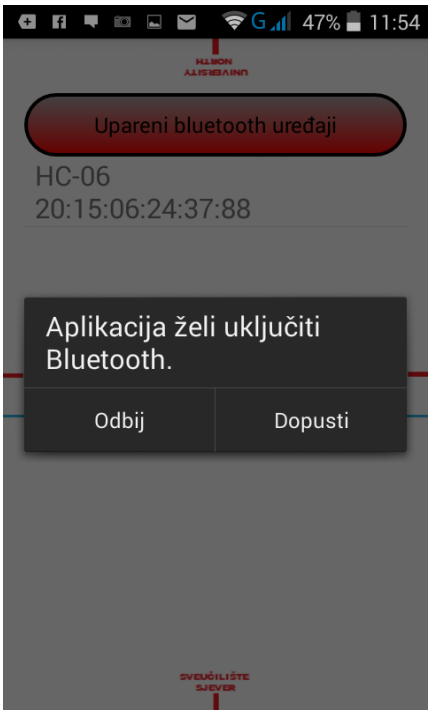
Nadalje, kako aplikacija koristi *Bluetooth* tehnologiju, najprije je potrebno u *manifest* datoteci dati dopuštenje aplikaciji kako bi mogla upravljati *Bluetooth* postavkama na mobilnom uređaju. To se čini slijedećom naredbom:

```
<uses-permissionandroid:name="android.permission.BLUETOOTH" />
```

Dalje u tekstu su prikazane i ukratko objašnjene funkcionalnosti prve aktivnosti *BluetoothActivity*, a nakon toga druge aktivnosti *kontrola*.

- **Prva aktivnost (*BluetoothActivity*)**

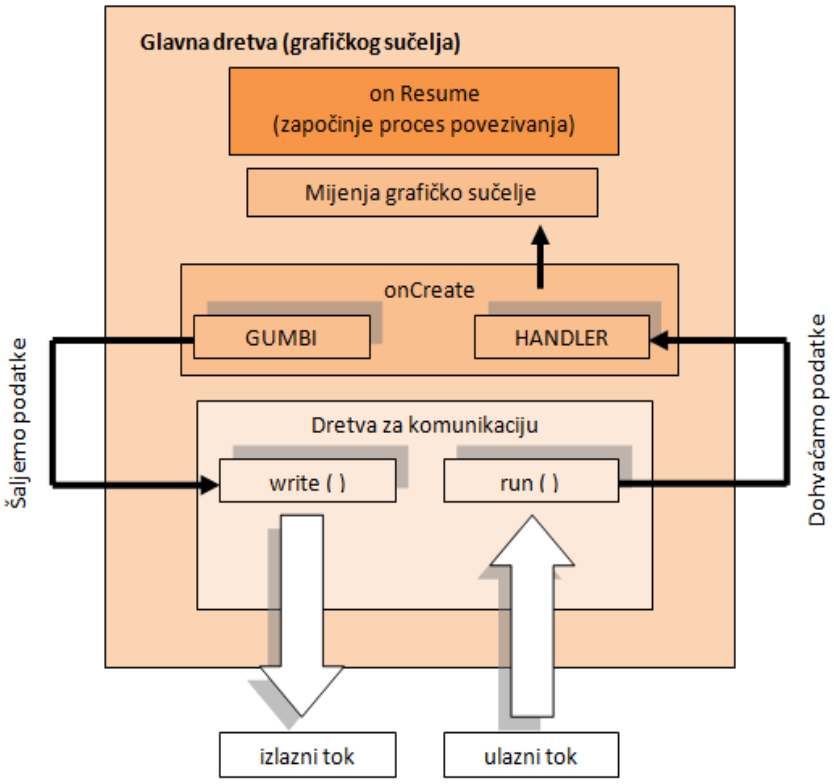
U prvoj aktivnosti, ukratko rečeno, imamo zaslon na kojem se nalazi gumb. Klikom na gumb pojavljuje se lista na kojoj se nalazi popis uparenih *Bluetooth* uređaja. Nakon odabira uređaja, prenose se informacije o odabranom uređaju (*MAC* adresa) u drugu aktivnost i započinje druga aktivnost *kontrola*. Isto tako, ukoliko *Bluetooth* na uređaju nije uključen, na ekranu se pojavljuje poruka da je za nastavak potrebno uključiti *Bluetooth* (slika 4-20.). Poruka se prikazuje tako dugo dok korisnik ne uključi *Bluetooth* jer je programski kod za prikaz poruke napisan u metodi *onResume*. Na drugoj aktivnosti započinje proces povezivanja , te ukoliko ne uspije, prebaci nas opet na početnu aktivnost. Sukladno tome, ukoliko se uređaji uspješno povežu, prebacuje nas na drugu aktivnost. U prilogu se nalazi dio programskog koda prve aktivnosti koji uključuje *Bluetooth* na mobilnom uređaju (prilog 3).



Slika 4-20. Primjer uključivanja Bluetootha

- **Druga aktivnost (*kontrola*)**

U *Android* dokumentaciji naglašeno je da se za ostvarivanje prijenosa i primanja podataka putem *Bluetooth* tehnologije treba kreirati zasebna dretva u kojoj se izvodi komunikacija [6]. Prema tome, druga aktivnost ima dvije dretve. U prvoj, glavnoj, odvija se prikaz grafičkog sučelja aktivnosti. U drugoj dretvi odvija se komunikacija sa *Bluetooth* modulom. Kreiramo ulazni i izlazni tok preko kojih se odvija komunikacija. Nadalje, potrebno je kreirati metodu koja nam služi za slanje podataka (*write*) i metodu koja nam služi za primanje podataka (*run*) No, prije svega, da bi se uređaji uopće mogli povezati, potrebno je kreirati *Socket*. *Socket* predstavlja krajnju točku komunikacije i da bi uređaji mogli komunicirati, moraju imati povezane *Sockete*. Proces kreiranja *Socketa*, stvaranja dretve i pokušaj slanja podataka odvija se u ciklusu, odnosno metodi *onResum*e (prilog 4). *OnCreate* ciklus možemo podijeliti na tri dijela. U prvom dijelu povezujemo vizualni dio aktivnosti sa programskim dijelom. U drugom dijelu omogućujemo slanje upravljačkih podataka preko gumba (*a, b, c, d, e, f*). Treći dio čini *Handler*. On nam zapravo služi za dohvat dolaznih podataka iz zasebne dretve i mijenjanje grafičkog sučelja glavne dretve. Koristimo ga iz razloga jer se grafičko sučelje ne može mijenjati iz zasebne dretve.



Slika 4-21. Najbitniji dijelovi druge aktivnosti

U primjeru niže prikazan je na koji način šaljemo znak *a* ili *b*. To činimo klikom na gumb pozivanjem metode *write* koja šalje podatke u izlazni tok. U prilogu se nalazi dio programskog koda u kojem kreiramo konstruktor dretve, ulazne i izlazne tokove te metode *write* i *run* (prilog 5).

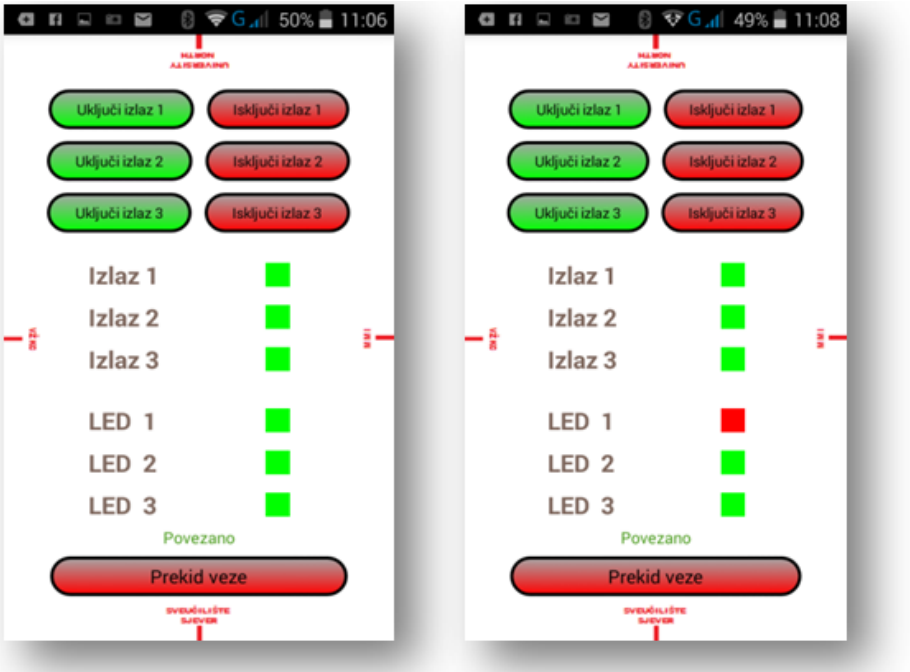
Primjer slanja znakova *a* ili *b* pomoću gumba

```
//postavljamo onClickListener
gumb1on.setOnClickListener(new OnClickListener()
{
        public void onClick(View v)
        {
              //preko zasebne dretve metodom write šaljemo znak a
              mConnectedThread.write("a");
         }
});

gumb1off.setOnClickListener(new OnClickListener()
{
         public void onClick(View v)
         {
              //šaljemo znak b
             mConnectedThread.write("b");
         }
});
```

Na primjeru niže opisan je *Handler* u kojem "filtriramo" podatke koji nas zanimaju i na temelju njih prikazujemo stanja izlaza *Arduina* i stanja *LED* dioda. Ukoliko podaci počinju sa znakom *, znamo da su to podaci koji nas zanimaju, prikazani na slici 4-5 u poglavlju 4.4. Pomoću metode *substring* odvajamo dijelove, odnosno izvlačimo, dolaznih podataka između znakova **:** i ovisno o tim podacima, prikazujemo stanje. Tako, primjerice, za *izlaz1* izvlačimo podatke između prvog i sedmog znaka. Podaci mogu biti *prvion* ili *prviof*. Kad je sadržaj podatka *prvion*, gumb je zelene boje, a kad je podatak *prviof*, gumb je crvene boje. U navedenom kodu, prikazan je primjer za „izvlačenje“ podatka za *izlaz1*, usporedba tog podatka i ovisno o podatku postavljanje zelene ili crvene boje gumba. Ista metoda "izvlačenja" podataka koristi se i za ostale izlaze i *LED* diode pa se dio koda koji provjerava sve izlaze i stanja *LED* dioda preskače i dolazi do kraja *Handler*-a gdje se dolazni podaci brišu za novo čitanje.

Primjer: Handler

```
handler= new Handler()
{
    public void handleMessage(android.os.Message msg)
    {
        if (msg.what == handlerstanje)
        {
            String readMessage = (String) msg.obj;
            dolaznipodaci.append(readMessage);
            int krajreda = dolaznipodaci.indexOf("#");
             //provjeri ima li podataka prije znaka #
            if (krajreda > 0)
            {
                // izvršava se kad je početni znak * (početak podataka)
                if (dolaznipodaci.charAt(0) == '*')
                {
                    // podaci o stanju izlaza1 (od 1. do 7. znaka)
                    String stizlaz1 = dolaznipodaci.substring(1,7);

                    //podaci o stanju izlaza2 (od 8. do 15. znaka)
                    String stizlaz2 = dolaznipodaci.substring(8, 15);

                    //podaci o stanju izlaza3 (od 16. do 23. znaka)
                    String stizlaz3 = dolaznipodaci.substring(16, 23);

                    //podaci o stanju LED1 (od 24. do 30. znaka
                    String stled1 = dolaznipodaci.substring(24, 30);

                    //podaci o stanju LED2 (od 31. do 37. znaka)
                    String stled2 = dolaznipodaci.substring(31, 37);

                    //podaci o stanju LED3 (od 38. do 44. znaka)
                    String stled3 = dolaznipodaci.substring(38, 44);

                     //usporedba podataka za stanje izlaza 1
                     if(stizlaz1.contentEquals("prvion") {       // uključen
                       stizl1.setBackgroundColor(Color.GREEN);  //zeleno
                     }
                     else if(stizlaz1.contentEquals("prviof")){  //isključen
                       stizl1.setBackgroundColor(Color.RED);    //crveno
                     }
                     // usporedba vrijednosti za ostale izlaze i LED diode
                     // po istom principu sa if else.....
                }
                //brišemo dolazne podatke
                dolaznipodaci.delete(0, dolaznipodaci.length());

            }
        }
    }
};
```



Slika 4-22. Izgled aplikacija u Android Studiu

4.7. Usporedba razvojnih sustava App Inventor i Android Studio

U ovom radu razvoj mobilne aplikacije izvršen je putem standardnog razvojnog sustava *Android Studio* te pomoću pojednostavljenog razvojnog sustava *App Inventor*. Ovdje se već naslućuje razlika između istih. Naime, *App Inventor* je jednostavan, namijenjen svima te omogućuje izgradnju funkcionalne aplikacije za *Android* uređaje. Sastoji se od dva dijela, *designer* i *block* dijela. Drugim riječima, nije potrebno pisati programski kod, već se aplikacije izrađuju po principu „*drag and drop*“, tj. poput slagalice. Iako jednostavan za korištenje, u nekim dijelovima ima svoja ograničenja. Primjerice, ne može se upravljati životnim ciklusom aktivnosti. U izrađenoj aplikaciji to ne igra bitnu ulogu, ali za imalo zahtjevnije aplikacije taj faktor zasigurno ima utjecaja. S druge strane, u usporedbi s njim, *Android Studio* je složeniji te uključuje *IDE* razvojno okruženje i *Android SDK* alate. Također, nudi niz mogućnosti za fleksibilnu izradu aplikacija. Za izradu aplikacija u *Android Studiu*, potrebno je poznavanje *Java* programskog jezika, što proces izrade aplikacije, u usporedbi sa *App Inventorom* čini puno složenijim. Stoga, zaključuje se kako je *App Inventor* idealan za realizaciju jednostavnijih aplikacija, dok *Android Studio* pruža mogućnost i izvedbu kompleksnih aplikacija uz uvjet dobrog poznavanja *Java* programskog jezika.

5. Kontrola preko web sučelja javno dostupnog poslužitelja

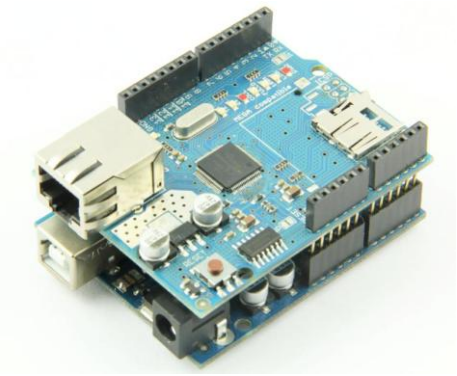
U ovom, drugom dijelu rada, objašnjena je mogućnost udaljene kontrole i nadzora uređaja preko web sučelja javno dostupnog poslužitelja. Kao i prvi dio, i ovaj dio je definiran na dva načina. Prvi je korištenjem poslužitelja i potrebnih modula na samom *Arduinu*, odnosno kad je *Arduino* poslužitelj i na njemu je web stranica (preko koje se vrši upravljanje i nadzor), dok drugi način obuhvaća korištenje javnog poslužitelja na kojeg se *Arduino* spaja kao klijent, šalje i s kojeg čita informacije. Dakako, na javnom poslužitelju nalazi se web stranica. U daljnjem tekstu je prvo objašnjen i opisan način kad je *Arduino* poslužitelj, a zatim je opisan drugi način, kad se *Arduino* pojavljuje u ulozi klijenta. Međutim, prije svega je potrebno opisati *Ethernet* modul, koji *Arduinu* služi za povezivanje na Internet.

5.1. Ethernet modul

Za realizaciju udaljene kontrole uređaja preko web stranice, korišten je *Ethernet Shield*, odnosno modul koji *Arduinu* omogućuje povezivanja na lokalnu (eng. *Local Area Network*) i javnu (eng. *Wide Area Network*) mrežu. Osnova mu je *Wiznet W5100* integrirani krug. Navedeni integrirani krug pruža mrežni stog koji podržava *TCP* (eng. *Transmission Control Protocol*) i *UDP* (eng. *User Datagram Protokol*) protokole. *Ethernet* modul spaja se na *Arduino* koristeći konektore koji se produžuju kroz *Ethernet* modul, što ne mijenja raspored pinova i omogućava spajanje još jednog modula na njega. *Ethernet* modul napajanje dobiva preko samog *Arduina*. Za povezivanje *Ethernet* modula na mrežu koristimo *UTP* (eng. *Unshielded Twisted Pair*) kabel sa standardnim *RJ-45* priključkom.

Arduino komunicira sa *Ethernet* modulom koristeći *SPI* (eng. *Serial Peripheral Interface*) sabirnicu. *SPI* sabirnica je sinkroni serijski podatkovni protokol, a radi na principu dvosmjerne komunikacije. Pinovi koji omogućuju tu komunikaciju su digitalni pinovi 10,11,12 i 13. Svaki pin ima određenu zadaću. Komunikacija se također odvija po principu gospodar (eng. *Master*) i rob (eng. *Slave*). *Arduino* je gospodar, a *Ethernet* modul rob. Pin 10- *SS* (eng. *Slave Select*) omogućuje *Arduinu* biranje roba ukoliko ima više spojenih modula na njega. Pin 11 još se naziva i *MOSI* (eng. *Master Out Slave In*) i preko tog pina *Arduino* šalje podatke *Ethernet* modulu. Pin 12, *MISO* (eng. *Master In Slave Out*) je pin sa kojeg *Arduino* prima podatke sa *Ethernet* modula. Pin 13, *SCK* (eng. *Serial Clock*) je pin koji daje impulse za sinkronizaciju prijenosa podataka. Kako bi se u programu mogle koristiti funkcije koje pruža *Ethernet* modul, potrebno je uključiti biblioteke <*Ethernet.h*> i <*SPI.h*>. Biblioteka <*Ethernet.h*> učita

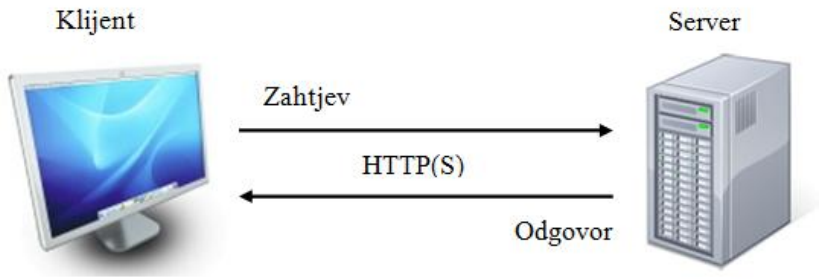
postavke mreže, a biblioteka <*SPI.h*> omogućuje komunikaciju s *SPI* uređajima, odnosno *Ethernet* modulom.



Slika 5-1. Ethernet Shield spojen na Arduino

5.2. Arduino kao poslužitelj

Arduino ima ulogu poslužitelja, a web preglednik je klijent. Klijent šalje zahtjev, poslužitelj obrađuje taj zahtjev i prosljeđuje podatke klijentu, što prikazuje slika 5-2. Opisane radnje izvršavaju se pomoću *HTTP* (eng. *HyperText Transfer Protocol*) protokola. Klijent šalje zahtjev poslužitelju u obliku metode zahtjeva, (*URI,* eng. *Uniform Resource Identifier*), verziju protokola, nakon čega slijedi izgled poruke koja sadržava modifikatore, klijentove informacije i moguće tijelo poruke [7]. Poslužitelj odgovara sa linijom statusa, uključujući verziju protokola, kod za uspjeh ili neuspjeh, nakon čega slijede informacije o entitetu i mogući sadržaj tijela entiteta. Svaki zahtjev ili odgovor završava praznom linijom. Nakon dostave odgovora, poslužitelj zatvara vezu (što *HTTP* čini *stateless* protokolom, tj. ne pamte se nikakve informacije o vezi između transakcija). *HTTP* koristi *TCP* protokol kojim se realizira pouzdana komunikacija [8]. Primjer klasičnog *HTTP* zahtjeva klijenta možemo vidjeti u prilogu (prilog 6). U našem slučaju klijent zatražuje stranicu pomoću *GET* metode, odnosno zahtjeva. Konkretno, kad se klijent spoji na *Arduino* poslužitelj, poslužitelj mu predaje stranicu napisanu *HTML* (eng. *HyperText Markup Language*) kodom. Unutar *HTML* koda stranice nalazi se forma za unos teksta, a njezina metoda je *GET*. Naredbe na koje *Arduino* reagira su, kao i u slučaju sa aplikacijama, slova *a, b, c, d, e* i *f*. Unosom jednih od tih slova i klikom na gumb *primjeni*, šalje se *GET* zahtjev na istu stranicu. *Arduino* obradi zahtjev i ovisno o zahtjevu, upravlja izlazima, odnosno *LED* diodama. U daljnjem tekstu nacrtan je dijagram koji prikazuje na koji način se odvija komunikacija i koju ulogu ima *Arduino* kao poslužitelj. Cijeli programski kod kad je *Arduino* poslužitelj, nalazi se u prilogu (prilog 7).



Slika 5-2. Model klijent - poslužitelj

5.2.1. Princip rada Arduino poslužitelja

Inicijalizacija mrežnih parametra

Arduino poslužitelj prvo inicijalizira mrežne parametre, *IP* (eng. *Internet Protocol*) i *MAC* adresu, te započinje sa radom.

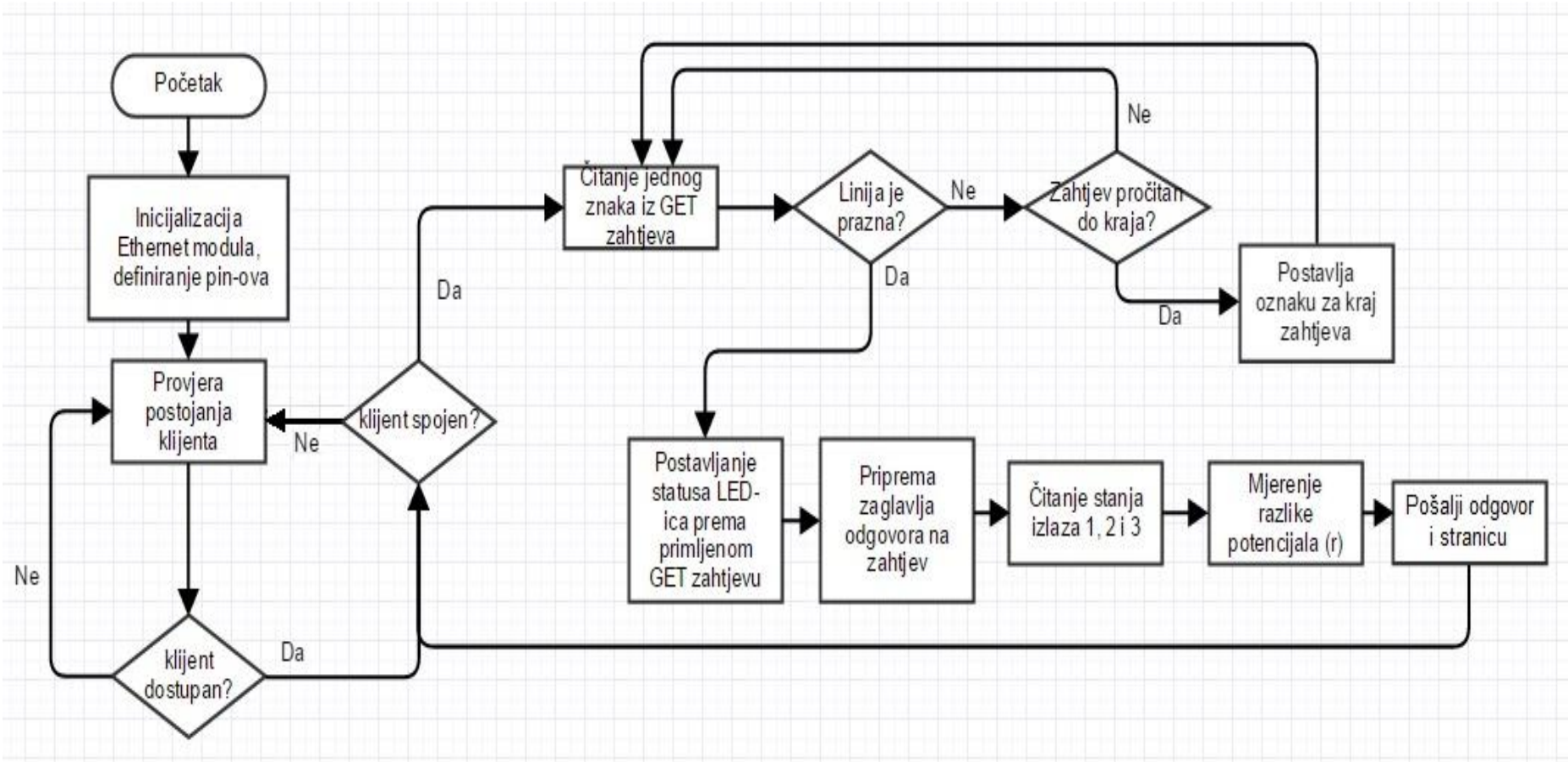
Čekanje klijenata

Slijedi provjera ima li dostupnih klijenata. Ako je klijent dostupan, odnosno, poslao *GET* zahtjev, *Arduino* ga prihvaća i kreće u njegovu obradu. Ako klijent nije dostupan, *Arduino* ponavlja traženje novog klijenta.

Obrada GET zahtjeva klijenta

Na početku obrade, prvo se provjerava da li je klijent poslao parametar „*LED*“ (definiran u formi), te da li su njegove vrijednosti *a, b, c, d, e* ili *f* (koje označavaju određenu naredbu koju *Arduino* mora obraditi). Kao i prije, naredbe *a, c* i *e* uključuju izlaze za *LED* diode (*izlaz1, izlaz2, izlaz3*), a naredbe *b, d* i *f* ih isključuju.

Slijedi priprema zaglavlja odgovora i priprema *HTML* koda web stranice koja će se vratiti kao odgovor. Kod pripreme *HTML* koda pozivaju se dvije funkcije. Jedna služi za očitavanje stanja izlaza *Arduina*, a druga za očitanje stanja *LED* dioda. Obje funkcije ispisuju *HTML* kod koji predstavlja trenutno stanje. Na slici 5-3. prikazan je tijek komunikacije *Arduino* poslužitelja sa klijentom (web preglednikom).



Slika 5-3. Princip rada Arduino poslužitelja

5.2.2. Web stranica

Za izradu web stranice korišteni su *HTML* i *CSS* (eng. *Cascading Style Sheets*). *HTML* upućuje na jezik za označavanje te mogućnost međusobnog povezivanja dokumenata hiper-poveznicama, odnosno *HTML* nam služi da definiramo sadržaj web stranice, dok je *CSS* tzv. „stilski jezik“ koji omogućava dodavanje različitih stilova *HTML* elementima, kao što su veličina i vrsta pisma, pozadinske boje, okviri, položaj elemenata, margine, itd. Dakle, *HTML* određuje način na koji će se prikazati određeni elementi na web stranici, dok *CSS* definira njezin vizualni izgled.

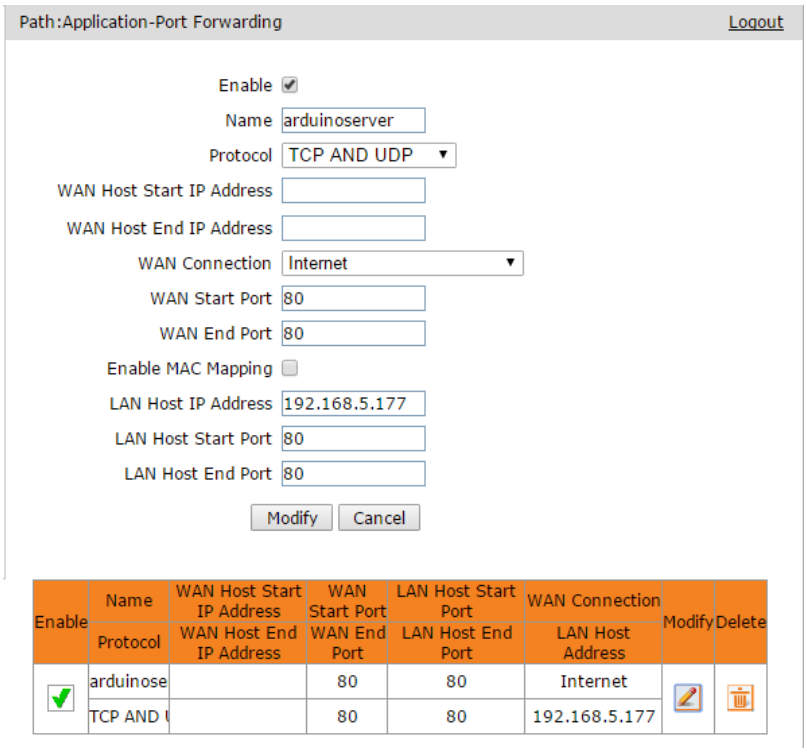
Slika 5-4. Web stranica na Arduinu

5.2.3. Pristup poslužitelju sa javne mreže (Internet)

Arduino, odnosno poslužitelj, nalazi se u lokalnoj (privatnoj) mreži. Kao što je spomenuto ranije, za identifikaciju na mreži dodijeljena mu je *MAC* i *IP* adresa. U većini slučajeva, *MAC* adresa nalazi se na samom *Ethernet* modulu, pa se nju uzima kao fiksnu. *IP* adresa koju dodjeljujemo poslužitelju treba biti unutar određenog raspona. Svaka *IP* adresa sastoji se od 32 bita, gdje je svakih 8 bitova odijeljeno točkom.

Prikazuju se u dekadskom obliku. Svaki dio odijeljen točkom može biti u rasponu od 0 do 255. Nadalje, svaka *IP* adresa ima dva dijela. To su mrežni dio (*Net ID*) koji određuje adresu mreže i dio koji određuje adresu uređaja unutar te mreže (*Host ID*). Svaka *IP* adresa dolazi uz pripadajuću masku podmreže (eng. *Subnet Mask*). Uz pomoć maske podmreže možemo razlučiti koji dio adrese je mrežni dio (*Net ID*), a koji dio za uređaje (*Host ID*). U našem slučaju maska podmreže je 255.255.255.0. (24 bita za adresu mreže, a 8 za uređaje), a adresa prvog uređaja 192.168.5.1. Primjenjući logičku operaciju I (eng. *AND*) nad ovim adresama, dolazimo do adrese mreže: 192.168.5.0. Drugim riječima, za adrese uređaja mogu se koristiti sve adrese između 192.168.5.1 i 192.168.5.254 (192.168.5.255 se koristi kao tzv. *broadcast* adresa, pa se ne može koristiti za adresu uređaja). Dodijeljena *IP* adresa poslužitelja je 192.168.5.177. Ovako konfiguriranom i povezanom na usmjeritelj, poslužitelju se može pristupiti unutar lokalne mreže. Unutar lokalne mreže moguće mu je pristupiti upisivanjem definirane *IP* adrese u web preglednik. Kako je port 80 standardan port kojeg *HTTP* protokol koristi za dohvat web stranica, nije ga potrebno navoditi kraj *IP* adrese. Portovima se zapravo definira o kojem procesu, odnosno servisu se radi. Kad bi, primjerice, u programu definirali da poslužitelj sluša na portu 8080, poslužitelju bi se pristupalo na način: 192.168.5.177:8080.

IP adrese se dijele na privatne i javne. Javne *IP* adrese su jedinstvene, globalne i standardizirane. Privatne adrese mogu biti duplicirane uz uvjet da se ne nalaze u istoj lokalnoj mreži. Podjela je nastala zbog manjka *IP* adresa. Prilikom „izlaska“ korisnika iz lokalne mreže na Internet, privatna *IP* adresa se pretvara u javnu *IP* adresu pomoću metoda *NAT* (eng. *Network Address Translation*) i *PAT* (eng. *Port Address Translation*) koje su implementirane u usmjeritelju. Znači, da bi se moglo pristupiti poslužitelju sa javne mreže, potrebno je znati javnu *IP* adresu usmjeritelja. Osim javne *IP* adrese usmjeritelja, koja nas dovodi do usmjeritelja, potrebno je konfigurirati usmjeritelj kako bi nas usmjerio na željeno računalo unutar lokalne mreže. Metodom kojom se to realizira, naziva se *Port Forwarding*. Konkretno, u toj metodi otvaramo port koji povezujemo sa *IP* adresom uređaja kojemu želimo pristupiti sa Interneta. U našem slučaju, otvaramo port 80 na kojem poslužitelj sluša i povezujemo ga sa *IP* adresom poslužitelja (192.168.5.177). Postavke su izvršene na modelu usmjeritelja *ZTE ZXV10 H201L* [9].



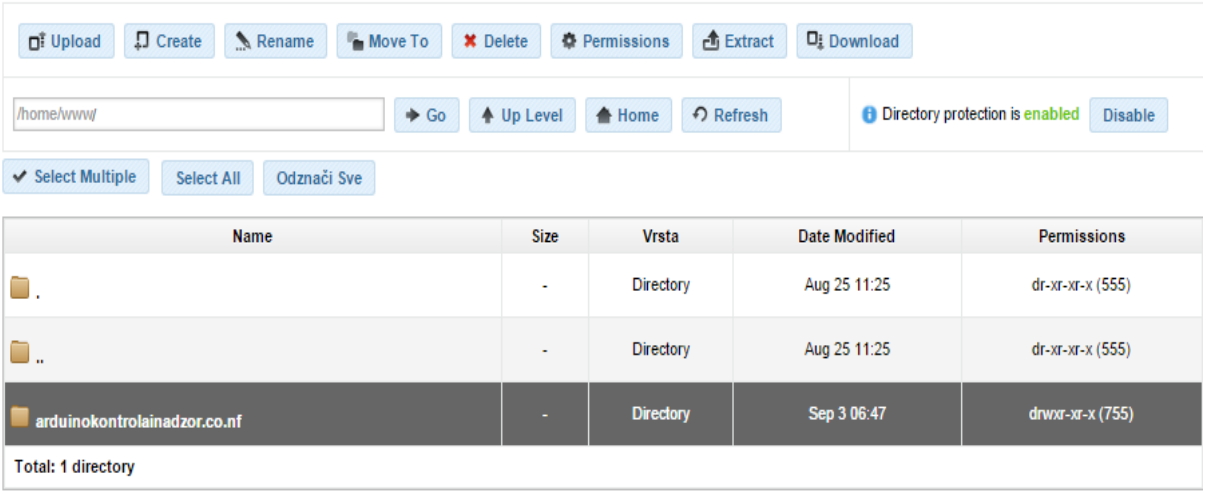
Slika 5-5. Otvaranja porta i povezivanje sa adresom poslužitelja

Iako *Ethernet* modul podržava *DHCP* (eng. *Dynamic Host Configuration Protocol*-automatski dodjeljuje mrežne parametre uređajima u mreži) protokol, ovdje je bitno da adresa poslužitelja bude statička, jer ukoliko bi se adresa mijenjala, adresa koju smo definirali u *Port Forwarding* metodi više ne bi bila ista i usmjeritelj ne bi imao kome proslijediti podatke. Nadalje, pojavljuju se dva problema koja je potrebno riješiti. Potrebno je riješiti problem dinamičke javne adrese usmjeritelja i umjesto *IP* adrese dodijeliti lako pamtljivo ime. Većina pružatelja usluga Interneta, usmjeritelju dodjeljuje dinamičku javnu adresu koja se s vremenom mijenja. Nadalje, puno je teže pamtiti brojeve (*IP* adrese), stoga je potrebno dati lako pamtljivo ime javnoj *IP* adresi usmjeritelja. Oba problema riješena su korištenjem servisa dinamičkog *DNS-a* (eng. *Domain Name System*). Naime, *DNS* čini hijerarhijski organizirana baza podataka distribuirana po poslužiteljima na Internetu [10], a svrha *DNS-a* je pridruživanje simboličke adrese *IP* adresi, odnosno lako pamtljivog imena (domena). Osim toga, pruža nam i uslugu ažuriranja javne *IP* adrese kod svake njezine promjene. U ovom radu korišten je besplatan servis dinamičkog *DNS-a*, a pružatelj te usluge je *NO-IP* [11]. Ime koje je dodijeljeno javnoj adresi usmjeritelja je *arduinowebserver.ddns.net*. Ukoliko želimo da se kod svake promjene javne adrese, ista ažurira sa navedenim imenom, potrebno je na bilo koji uređaj unutar mreže instalirati program koji nam to omogućuje. Moguće ga je preuzeti sa službene stranice *NO-IP*-

a. Time smo ostvarili pristup poslužitelju (koji se nalazi u lokalnoj mrži) sa javne mreže preko lako pamtljivog imena.

5.3. Arduino kao klijent

U ovom dijelu upravljanje i nadzor uređaja realizira se preko web stranice koja se nalazi na zasebnom serveru. Za realizaciju zasebnog servera korištena je usluga besplatnog *web hosting-a* [12]. Izradom računa na spomenutom servisu, kreiramo web adresu poslužitelja te nam je dodijeljen određen dio podatkovnog prostora u kojeg možemo pohranjivati resurse, u ovom slučaju web stranicu i popratne datoteke. Web adresa kojom se može pristupiti web stranici je [13]. Nadalje, web stranicu i ostale datoteke stavljamo u automatski kreiranu mapu *arduinokontrolainadzor.co.nf* (slika 5-6.).



Slika 5-6. Kreirana mapa - arduinokontrolainadzor.co.nf

Za realizaciju upravljanja i prikaz stanja putem stranice, između ostalog, korišten je *PHP* (eng. *Hypertext Preprocessor*) programski jezik. Najprije će ukratko biti opisan način upravljanja i nadzora uređaja preko web stranice, a nakon toga osnove programskog jezika *PHP* sa opisom pojedinih dijelova koda. Naime, preko stranice se vrši upravljanje izlazima *Arduina* na način da se pomoću linkova prosljeđuju varijable, odnosno vrijednosti tih varijabli, *PHP* skriptama koje se nalaze na istom poslužitelju. Svaka vrijednost linka (atribut) je različita i jednoznačno definira što *Arduino* mora činiti ovisno o vrijednosti. Postoje tri skripte, za svaki izlaz jedna. Skripta dohvaća vrijednost i zapisuje tu vrijednost u tekstualnu datoteku. *Arduino*, u ulozi klijenta, konstantno se vrti u petlji, odnosno konstantno šalje zahtjeve i zahtjeva sadržaje tih datoteke, te isto tako šalje stanja svojih izlaza i stanja *LED* dioda. Server mu vraća odgovor i nakon odgovora pruža mu sadržaj tražene datoteke. Stanja se pomoću *PHP* skripte dohvaćaju

i spremaju u tekstualne datoteke. Te vrijednosti se sa web stranice dohvaćaju, također pomoću *PHP*-a, i na temelju njih prikazuju stanja izlaza *LED* dioda i stanja izlaza *Arduina*.

5.3.1. PHP programski jezik

PHP je programski jezik, raširen i besplatan a služi za izradu dinamičkih i interaktivnih web stranica. Drugim riječima, to je skriptni jezik za programiranje na strani poslužitelja. *PHP* programski kod ugrađuje se u tekst *HTML* dokumenta, a može biti napisan zasebno kao skripta. Izlazak iz *HTML*-a i ulaz u *PHP* kod prikazan je na primjeru ispod.

```
<?php //početak  php skritpe
 //php kod
?> //završetak  php skripte
```

5.3.2. Funkcije pojedinih dijelova koda

U mapi *arduinokontrolainadzor.co.nf* nalazi se stranica (*index.php)*, te *PHP* skripte naziva *izlaz1.php, izlaz2.php, izlaz3.php* i *dolaznipodaci.php*. Zadaće koje obavljaju navedene skripte te programski dio *Arduina* kad se ponaša kao klijent, bit će objašnjene u daljnjem tekstu.

Kao što je već spomenuto, sa web stranice se pomoću linkova prosljeđuju određene vrijednosti u *PHP* skripte. Točnije, prosljeđujemo ih u *URL*-u (eng. *Uniform Resource Locator). URL* je standardna adresa koja se koristi za pronalaženje određenog resursa na Internetu. Pritom, varijable se dodaju u *URL*-u na kraj, npr. *http://mojastranica.hr/mojaphpstranica.php?var=1*. U ovom linku je nakon stranice dodan upitnik, koji govori pregledniku da se iza njega nalazi varijabla te nakon njega *ime_varijable=vrijednost*. Primjer opisanih dijelova koda vezan je za *izlaz1*. Na web stranici se za *izlaz1* prosljeđuju znakovi * i %, ovisno želimo li uključiti ili isključiti navedeni izlaz. Znak koji označava da je potrebno uključiti *izlaz1* je *, a za isključiti %. Za *izlaz2*, znak > označava da je potrebno uključiti izlaz, a znak < isključiti. Za *izlaz3* za uključiti se koristi znak ^, a za isključiti *!*.Ovaj izbor znakova je naizgled čudan, ali razlog njihovog korištenja bit će spomenut kod opisa programa na samom *Arduinu*.

Primjer koda koji se odnosi na dio za prosljeđivanje vrijednosti pomoću linkova (izlaz1)

```
<a href ="izlaz1.php?izlazprvi=*"><button>Uključi izlaz 1</button></a>
<a href ="izlaz1.php?izlazprvi=%"><button>Isključi izlaz 1</button></a>
```

U skripti *izlaz1.php* se pomoću *PHP* metode *GET* dohvaća vrijednost varijable *izlazprvi* sa *index.php* stranice. Vrijednost se sprema u varijablu *$izl1*. Naime, znak *$* ispred naziva je obavezan i on predstavlja sintaksu za kreiranje varijable u *PHP* programskom jeziku. Nakon dohvata varijable, spremamo je u tekstualnu datoteku imena *izlaz1.txt*, pomoću funkcija *fopen, fwrite,* i *fclose*. Funkcija *fopen* upotrebljava se i za otvaranje i za kreiranje datoteke. Parametri koje prosljeđujemo unutar zagrada *fopen* funkcije su ime datoteke koju želimo kreirati (*izlaz1.txt*) i parametar *w* (eng. *write*) koji sugerira da želimo pisati u datoteku. Nakon otvaranja tekstualne datoteke *izlaz1.txt*, pomoću funkcije *fwrite* zapisujemo vrijednost dohvaćene varijable u datoteku. Nakon otvaranja i pisanja, datoteku je potrebno zatvoriti funkcijom *fclose*. Nakon toga, vraćamo se na glavnu stranicu pomoću naredbe *header*. Isti princip koristi se i za ostale izlaze, samo se prosljeđuju različite vrijednosti varijabli koje se spremaju u zasebne tekstualne datoteke.

Primjer: PHP skripta izlaz1.php

```
<?php
// dohvaća vrijednost varijable izlazprvi
$izl1 = $_GET["izlazprvi"];

//kreira, otvara, zapisuje vrijednost u tekstualnu datoteku
// i zatvara datoteku
$tekstdatoteka1 = fopen("izlaz1.txt", 'w');
fwrite($tekstdatoteka1, $izl1);
fclose($tekstdatoteka1);

//vraća se na index.php
header("Location:index.php");
```

Da bi *Arduino* mogao pročitati gore navedene podatke, mora ih na neki način dohvatiti sa servera. To čini pomoću *HTTP* protokola i metode *GET*. Odnosno, šalje zahtjev za čitanje tekstualne datoteke i na temelju vrijednosti u datoteci, upravlja izlazom.

Klijent (*Arduino*) se najprije pokušava spojiti na poslužitelj. Nakon toga, šalje *HTTP* zahtjev. U primjeru, šalje zahtjev kojim zahtjeva sadržaj tekstualne datoteke *izlaz1.txt*. Kao što je gore navedeno, u toj datoteci nalazi se znak * ili znak %. U zahtjevu je najprije navedena metoda (*GET*), nakon čega slijedi *URL* koji nas dovodi do traženog resursa. Potrebno je navesti

putanju do datoteke i naziv datoteke čiji sadržaj se zahtijeva. Nakon poslanog zahtjeva, poslužitelj mu šalje odgovor i sadržaj tražene datoteke. *Arduino* čita odgovor, a nakon odgovora čita sadržaj datoteke.

```
HTTP/1.1 200 OK
Date: Sun, 06 Sep 2015 21:36:50 GMT
Server: Apache
Last-Modified: Sun, 06 Sep 2015 21:14:59 GMT
ETag: "18d6cc9-1-51f1aa01edc99"
Accept-Ranges: bytes
Content-Length: 1
Connection: close
Content-Type: text/plain
```

Slika 5-7. HTTP odgovor poslužitelja

Iz slike 5-7 vidimo koji odgovor poslužitelj šalje *Arduinu*. Možemo vidjeti da se nigdje u odgovoru poslužitelja kojeg *Arduino* čita ne nalaze znakovi (*,%), a u programu mu je definirano da upravlja *izlazom1* jedino ako pročita jedan od navedenih znakova (*,%). Bilo je problema sa čitanjem podataka nakon završetka odgovora, odnosno nakon prazne linije (\n), kao što je to bio slučaj kad je *Arduino* poslužitelj, pa se iz tog razloga koriste posebni znakovi (*, %, i sl.) koji jednoznačno definiraju što *Arduino* mora činiti kad pročita jedan od tih znakova. Na ovaj način „zanemarujemo“ podatke, odnosno odgovor kojeg nam poslužitelj šalje, i „filtriramo“ samo one podatke koji nam trebaju, a to su podaci u traženoj datoteci (* ili %).. Ista metoda koristi se za upravljanje izlazima 2 i 3, samo što *Arduino* zahtjeva sadržaj datoteka *izlaz2.txt* i *izlaz3.txt* u kojima se nalaze različiti podaci (*izlaz2*- <, >, *izlaz3*- ^, *!*).

Programski kod za upravljanje prvim izlazom (izlaz1)

```
//čitanje podataka za izlaz 1
if (klijent.connect(server, 80))
{
   Serial.println("povezano 1");
   //zahtjev za datoteku izlaz1.txt
   klijent.println("GET http://arduinokontrolainadzor.co.nf/izlaz1.txt HTTP/1.1");
   klijent.println("Host: arduinokontrolainadzor.co.nf");
   klijent.println("Connection: close");
   //kraj zahtjeva-prazna linija
   klijent.println();

    //klijent se smatra povezanim, unatoč tome što
    //je veza zatvorena (Connection:close),
    //ukoliko ima još podataka za čitanje
    while (klijent.connected())
    {
       // vraća broj bajtova dostupan za čitanje (od poslužitelja)
       while (klijent.available())
       {
         // ukoliko ima podataka za čitanje, čitaj znak po znak
         // i spremaj u varijablu ch1
         char ch1 = klijent.read();
         //Serial.print(ch1);--> za prikaz pročitanih podataka

            // kad je ch1 jednako znaku *, uključi izlaz 1
            if (ch1 == '*') {
              digitalWrite(izlaz1, HIGH);
              Serial.println("izlaz 1 ukljucen");
            }

            // ako je ch1 jednako znaku %, isključi izlaz 1
            elseif (ch1 == '%'){
              digitalWrite(izlaz1, LOW);
              Serial.println("izlaz 1 iskljucen");
            }
       }
     }
  //odspajanje
  klijent.stop();
 delay(300);
}
else {  Serial.println("citanje izlaz1-- citanje nije uspjelo"); }
```

U sljedećem tekstu opisan je način slanja podataka sa *Arduina* i dohvat tih podataka na poslužitelju. Naime, *Arduino* čita stanja svojih izlaza i sprema ih u varijable (*stizl1, stizl2, stizl3*). Također, mjeri razliku napona i vrijednosti sprema u varijable (*stled1, stled2, stled3*). Te vrijednosti šalje na poslužitelj u datoteku *dolaznipodaci.php*. Vrijednosti se šalju u URL-u, a svaka vrijednost mora biti odvojena znakom &. Vrijednosti varijabli su 1 (za uključeno

stanje) ili 0 (za isključeno stanje). Cijeli programski kod *Arduina* kad je klijent nalazi se u prilogu (prilog 8).

Primjer slanja podataka sa Arduina na poslužitelj

```
//slanje stanja izlaza i LED dioda
if (klijent.connect(server, 80))
{
     Serial.println("slanje stanja ");
     //slanje podataka HTTP protokolom u datoteku dolaznipodaci.php
     klijent.print("GEThttp://arduinokontrolainadzor.co.nf/dolaznipodaci.php?");
     klijent.print("izl1=");
     klijent.print(stizl1);
     klijent.print("&izl2=");
     klijent.print(stizl2);
     klijent.print("&izl3=");
     klijent.print(stizl3);
     klijent.print("&led1=");
     klijent.print(stled1);
     klijent.print("&led2=");
     klijent.print(stled2);
     klijent.print("&led3=");
     klijent.print(stled3);
     klijent.println(" HTTP/1.1");
     klijent.println("Host: arduinokontrolainadzor.co.nf" );
     klijent.println("Connection: close" );
     klijent.println();
     klijent.stop();
     delay(300);
}

else  { Serial.println("podaci nisu poslani");  }
```

Dohvat podataka koje *Arduino* šalje, vrši se u *PHP* skripti *dolaznipodaci.php*. Dohvaćeni podaci, svaki posebno, sprema se u zasebnu tekstualnu datoteku. Prikazan je primjer dohvata vrijednosti stanja za *izlaz1*.

Vrijednost se dohvaća, sprema u varijablu koja se zapisuje u tekstualnu datoteku *stanjeizlaz1.txt*. Po istom principu, dohvaćaju se i ostali podaci, odnosno stanja.

Primjer dohvata podataka za izlaz1

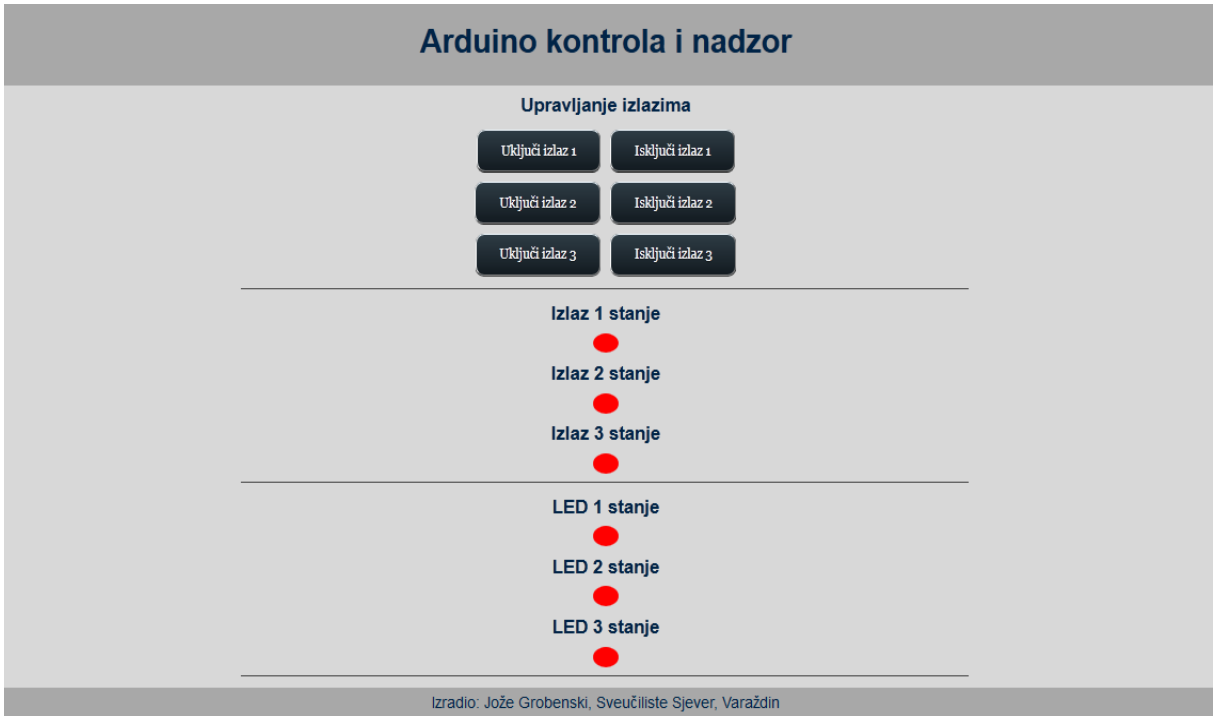
```
$stanjeizlaz1=$_GET["izl1"];
$otvori1=fopen("stanjeizlaz1.txt", 'w');
fwrite($otvori1, $stanjeizlaz1);
fclose($otvori1);
```

Da bi se na stranici prikazivala stanja, u *HTML* kod umetnuta je *PHP* skripta koja dohvaća stanja, odnosno sadržaj tekstualnih datoteka i, ovisno o sadržaju tih datoteka, ispisuje *HTML* kod kojim se prikazuju njihova stanja. Funkcija *echo* služi za ispis podataka, u ovom slučaju *HTML* koda. Funkcijom *file_get_contents* dohvaća se sadržaj određene datoteke. Po istom principu prikazuju se stanja za ostale izlaze i *LED* diode.

Primjer koda za prikaz stanja izlaza1 na web stranici

```
echo'<h3>Izlaz 1 stanje</h3>';

$stizl1=file_get_contents("stanjeizlaz1.txt");
// uključen
if($stizl1=="1"){
  echo'<div class="zeleni"></div>';// ispisuje zeleni gumb
}
// isključen
elseif($stizl1=="0"){
  echo '<div class="crveni"></div>'; // ispisuje crveni gumb
}
```

Slika 5-8. prikazuje izgled web stranice na zasebnom serveru. Programski kod stranice nalazi se u prilogu (prilog 9).



Slika 5-8. Web stranica na zasebnom serveru

5.4. Usporedba Arduina u ulozi poslužitelja i u ulozi klijenta

Kad je *Arduino* poslužitelj, sav programski kod nalazi se na njemu. Kako bi mu se omogućio pristup sa javne mreže, potrebno je konfigurirati usmjeritelj. Vidljivo je kroz postupak realizacije poslužitelja koje postupke je za to potrebno provesti. Nadalje, kako bi se poslužitelju moglo pristupiti kod promjene javne adrese usmjeritelja, vidimo kako je potrebno na bilo kojem uređaju (računalu) povezanom na tu istu mrežu instalirati program koji vrši ažuriranje dinamičke *IP* adrese sa dodijeljenom domenom. Idealan slučaj bio bi kad bi usmjeritelj podržavao jednu od besplatnih usluga dinamičkog *DNS*-a te ne bi bilo potrebe za programom koji vrši ažuriranje *IP* adrese.

Nadalje, idealno bi bilo kad bi se koristio *Ethernet* modul koji ima mogućnost napajanja preko *UTP* kabla, odnosno *PoE* (eng. *Power over Ethernet*). Korišteni modul nema tu opciju, a time bi se uklonila potreba zasebnog napajanja. Što se odziva tiče, pošto se poslužitelj izvršava na samom *Arduinu*, odziv je veoma dobar. Napisan programski kod za poslužitelj zauzima puno radne memorije samog *Arduina*. Iz toga se zaključuje da je realizacija *Arduina* kao poslužitelja idealna za manje i ne zahtjevne projekte upravljanja. Ukoliko bi namjena poslužitelja bila obrada ili spremanje većih količina podataka, unatoč mogućnosti korištenja *SD* (eng. *Secure Digital*) kartice, sustav je i dalje poprilično limitiran po pitanju memorije. Upravo u tom segmentu, drugi način opisan u radu, kad je *Arduino* klijent, ističe prednosti. *Arduino* kao klijent, samo šalje i čita podatke sa poslužitelja. Iz toga se zaključuje kako se veći dio posla obavlja na poslužitelju. Osim toga, postoji nekoliko mogućosti pohranjivanja podataka i način njihove obrade. U ovom primjeru su tekstualne datoteke poslužile svrsi, no ukoliko bi se radilo o podacima koji se žele periodički, odnosno vremenski pregledavati, postoji mogućnost kreiranja baze podataka, spremanja u bazu i po potrebi njihov prikaz na stranici. Odziv je očekivano slabiji kad *Arduino* ima ulogu klijenta, jer spajanje na zaseban poslužitelj iziskuje određeno vrijeme.

6. Optimizacija projektiranog sustava

U realiziranom sustavu podaci o stanjima *LED* dioda i izlaza *Arduina* šalju se stalno. U varijanti sa upravljanjem putem *Bluetooth-a*, podaci o stanjima generiraju se u funkciji *citajstanje* koja se konstantno poziva iz glavne petlje *loop*. Ti podaci se konstantno šalju aplikaciji. S druge strane, kad se upravljanje vrši preko web stranice, automatskim osvježavanjem web stranice, podaci se konstantno šalju što stvara velik i nepotreban mrežni promet. U praksi je poželjno optimizirati slanje podataka i smanjiti količinu prometa.

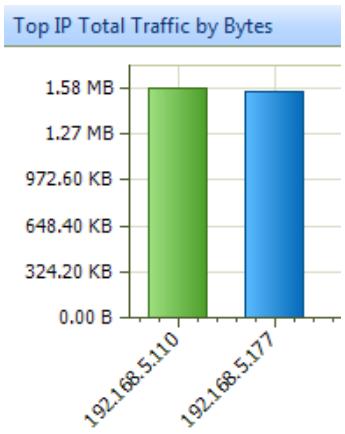
6.1. Optimizacija prometa putem Bluetooth komunikacije

Kod verzije sa *Bluetooth*-om, problem konstantnog slanja podataka je riješen na način da se podaci šalju samo kod promjene stanja, što je učinjeno izmjenom same funkcije *citajstanje*. Uvedene su dvije pomoćne varijable. Jedna nam služi za spremanje starih stanja, a druga za spremanje trenutnih stanja. Te dvije varijable se međusobno uspoređuju i podaci se ispisuju, odnosno šalju aplikaciji jedino ako su vrijednosti varijabla različite, odnosno promijenilo se stanje u odnosu na prijašnje stanje. Izmijenjena funkcija *citajstanje* nalazi se u prilogu rada (prilog 10).

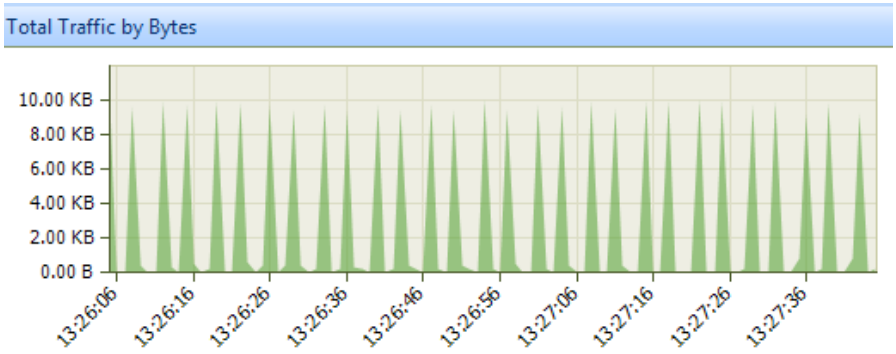
6.2. Optimizacija kad je Arduino poslužitelj

Mjerenjem prometa koji se generira kad je *Arduino* poslužitelj, utvrđeno je da se najviše prometa troši na automatsko osvježavanje web stranice. Konkretno, kada je klijent (web preglednik) spojen na *Arduino* poslužitelj, stranica se osvježava svake tri sekunde kako bi se prikazivala stanja. Sve što web preglednik prikazuje, zapravo *Arduino* generira, odnosno, *Arduino* ispisuje pregledniku *HTML* kod kojega preglednik prikazuje u obliku web stranice. Iz toga se zaključuje kako je količina podataka koje generira web preglednik i *Arduino* gotovo identična. Provedeno je mjerenje potrošnje mrežnog prometa u trajanju od 10 minuta, a prikazano je na slici 6-1. Na slici je prikazan generirani mrežni promet i *IP* adrese uređaja koje generiraju taj mrežni promet. Adresa 192.168.5.110 je adresa računala koje je spojeno na *Arduino* poslužitelj, odnosno na tom računalu web preglednik prikazuje web stranicu koju *Arduino* generira. Adresa 192.168.5.177 je adresa *Arduino* poslužitelja. Iako se radi o ne zahtjevnoj stranici sa malo *HTML* koda, podaci koji se generiraju u periodu od 10 minuta, sa

stalnim osvježavanjem stranice iznose nešto više od 3 MB (3,15 MB), što i nije malo (ukoliko bi uzeli duži vremenski period). Slika 6-2. prikazuje dio generiranog prometa u vremenskim intervalima od 10 sekundi. Vidljivo je kako se promet generira otprilike tri puta unutar 10 sekundi, što zapravo predstavlja automatsko osvježavanje stranice svake tri sekunde.

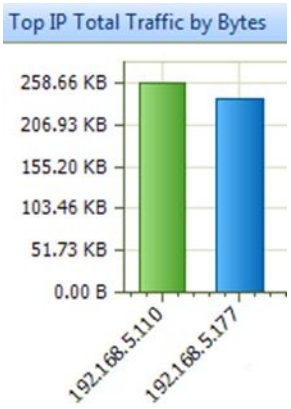


Slika 6-1. količina generiranih podataka kod ne optimizirane verzije

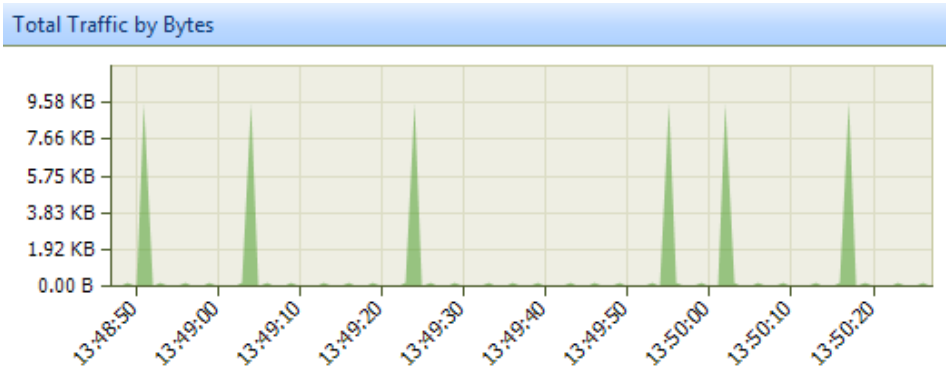


Slika 6-2. generirani promet u intervalima od 10 sekundi (osvježavanje svake 3 sekunde)

Optimizacija je provedena na način da je unutar *HTML* koda web stranice izostavljen dio za automatsko osvježavanje stranice. Osvježavanje, odnosno provjera stanja se može vršiti putem gumba *Provjeri Stanje* na zahtjev, odnosno po želji korisnika. Za tu verziju također je provedeno 10- minutno mjerenje prometa kojim je utvrđeno kako se promet uvelike smanjio što prikazuje slika 6-3.. Generirani promet iznosi otprilike 0,5 MB, što je višestruko manje od prijašnje verzije sa automatskim osvježavanjem. Na slici 6-4. vidljivo je da se promet generira samo kod zahtjeva korisnika, a ne svake tri sekunde.



Slika 6-3. potrošnja prometa bez automatskog osvježavanja stranice



Slika 6-4. generirani promet u intervalima od 10 sekundi (na zahtjev korisnika)

6.3. Optimizacija kad je Arduino klijent

U ne optimiziranoj verziji, podaci o stanjima izlaza i *LED* dioda slali su se konstantno, dok se za upravljanje izlazima za svaki izlaz zasebno slao zahtjev za određenom tekstualnom datotekom. Ovdje je optimizacija provedena na način da se stanja šalju samo kad se dogodi promjena. Nadalje, *Arduino* ne upravlja izlazima tako što šalje za svaki izlaz zaseban zahtjev, već sve čini sa jednim zahtjevom, tj, čita tri znaka koja sugeriraju što je potrebno učiniti sa izlazima. *Arduino* programski kod za optimiziranu verziju *Arduino* klijenta nalazi se u prilogu dokumenta (prilog 11). Sukladno promjeni *Arduino* programa, na serveru su načinjene određene promjene. Načinjene su dvije nove *PHP* skripte, dok se ostale koje su se koristile u prijašnjoj ne optimiziranoj verziji ne koriste, te je glavna stranica *index.php* u nekim dijelovima promijenjena (izmjene *index.php* stranice nalaze se u prilogu- prilog 12) . Također, umjesto automatskog osvježavanja stranice, stavljen je gumb *Dohvati stanja*. *PHP* skripte za

optimiziranu verziju nalaze se u prilogu dokumenta (prilog 13, 14). U prvoj skripti naziva *postavi.php*, dohvaćaju se stanja koja šalje *Arduino* i spremaju u tekstualnu datoteku *stanje.txt*. Također, u *postavi.php* dohvaćaju se parametri sa web stranice (*index.php*) koji sugeriraju da li je potrebno uključiti određeni izlaz i te vrijednosti spremaju se u tekstualnu datoteku *stanje0.txt*. U *PHP* skripti *dohvati.php* dohvaćaju se stanja iz tekstualne datoteke *stanje0.txt* i ovisno o vrijednostima ispisuju se određeni znakovi koje *Arduino* čita i na temelju njih upravlja izlazima.

Provedeno je mjerenje potrošnje prometa sa starom, ne optimiziranom, i novom verzijom. Na „upravljačkoj ploči“ servera prikazuju se podaci koji govore kolika je potrošnja prometa na serveru te su se ti podaci uzimali kao mjera potrošnje prometa. Mjerenje se vršilo u periodu od 10 minuta. Početno stanje prometa iznosi 77,82 MB. To je sveukupan dosadašnji promet na serveru. Iz tablice se vidi da je potrošnja prometa znatno snižena kod optimizirane verzije.

	Početno stanje	Završno stanje	Ukupan promet
Stara verzija	77.82 MB	79,76 MB	1,94 MB
Optimizirana verzija	79,76 MB	80,84 MB	0,72 MB

Tablica 6-1. Potrošnja prometa prije i poslije optimizacije

7. Rekapitulacija troškova

U svrhu razvoja *Arduino* sustava bilo je potrebno koristiti jezgru uređaja te dodatne module. Iz tog su razloga na ovaj rad utrošeni određeni financijski resursi. Naime, što se tiče troškova korištenih modula, postojale su dvije opcije. Koristiti hrvatsko tržište ili kupnju preko Interneta (npr. *eBay*). Pritom se, kao glavni uvjet, uzela u obzir prvenstveno cijena. Realno gledano, hrvatsko tržište nudi *Arduino* uređaje po višoj cijeni, nego je cijena istog na Internetu, tj. *eBay*-u, koji pruža online uslugu kupnje i prodaje. Naravno, pritom treba uzeti u obzir činjenicu da se putem online kupnje dobio uređaj kineskih proizvođača (kopija) čija je kvaliteta eventualno nešto manja. Također, isti je princip i sa *Ethernet* modulom te *HC-06 Bluetooth* modulom. Za rad je korišten originalan *Arduino*, kupljen na hrvatskom tržištu, dok su *Ethernet* modul i *HC-06 Bluetooth* modul kupljeni putem Interneta. U tabeli 5-1. su prikazane i uspoređene cijene na hrvatskom tržištu te putem online kupnje. Vidljive su već spomenute značajne razlike u cijenama, ovisno o tržištu kupnje.

TRŽIŠTE / UREĐAJ	HRVATSKO TRŽIŠTE (HRK)	EBAY (HRK)
Arduino Uno R3	133,38	24,60
Ethernet modul	173,63	63,69
HC-06 Bluetooth modul	99,00	27,10

Tablica 7-1. Usporedba cijena uređaja

Konkretno, kad se zbroje sredstva utrošena na kupnju potrebnih uređaja, stanje je sljedeće. Tabela 5-2. prikazuje stvarno utrošena sredstva za kupnju uređaja, što u konačnici iznosi ukupno 224, 17 kn.

UREĐAJ	CIJENA (HRK)
Arduino Uno R3	133,38
Ethernet modul	63,69
HC-06 Bluetooth modul	27,10
UKUPNO	**224,17**

Tablica 7-2. Troškovi kupnje uređaja

8. Primjena realiziranog sustava u praksi

Iz svega opisanog u radu vidimo kako je moguće realizirati udaljenu kontrolu i nadzor uređaja baziranih na *Arduino* platformi na više načina. Konkretno, u našem slučaju putem bežične *Bluetooth* tehnologije i web sučelja javno dostupnog poslužitelja (web stranice). Implementirani uređaj nema konkretne i značajne funkcionalnosti, već samo služi za demonstrativne svrhe. Ono bitno je realizacija mogućnosti udaljene kontrole samog *Arduina*, odnosno njegovih izlaza i mogućnost dobivanja povratnih informacija sa njega samog. Na temelju toga, mogu se realizirati sustavi upravljanja koji obavljaju konkretnu zadaću.

Jakost struje i sam napon kojeg *Arduino* platforma isporučuje ne mogu se direktno koristiti za kontrolu većih potrošača, odnosno uređaja. Ovaj nedostatak lako se uklanja zbog izrazito velike modularnosti *Arduino* platforme. Tako se, primjerice, može uz neznatne financijske troškove kupiti modul opremljen relejima (eng. *Relay board*). Modul s relejom omogućava kontrolu trošila koji zahtijevaju veću struju i napon, bez obzira rade li na istosmjernim ili izmjeničnim veličinama. Relej možemo aktivirati signalom kojega šaljemo preko *Arduina* (5 V). Relej ovdje zapravo ima ulogu prekidača, kojim možemo zatvoriti ili otvoriti strujni krug potrošača. Potrošač mora imati zaseban izvor napajanja. Prikazani modul na slici 8-1. ima četiri releja, što znači da se njime mogu kontrolirati četiri trošila odjednom (za prikazani relej, max. vrijednosti za izmjenične veličine su 10 A, 250 V AC, a za istosmjerne 10 A i 30 V DC). Ista funkcija upravljanja kao i relejima, može se realizirati korištenjem tranzistora, u kojem tranzistor ima ulogu tzv. tranzistorske sklopke.

Tako se, primjerice, korištene *LED* diode u radu mogu zamijeniti relejima i pomoću releja možemo vršiti upravljanje raznih uređaja. U spomenutoj metodi kontrole uređaja preko releja ili tranzistora, nije potrebno čitati stanja tog uređaja kao što je to provedeno u ovom radu mjerenjem napona, već se to može realizirati na druge načine. Konkretno, za to možemo koristiti razne senzore koji su dostupni za *Arduino* platformu. Samo neki od najupotrebljavanijih i najkorištenijih senzora su senzori temperature, tlaka , vlage, svjetline, pokreta, razine tekućine i sl. U većini slučajeva, senzori se priključuju na analogne ulaze *Arduina*.

Navedeni primjeri upravljanja preko releja ili tranzistora i očitanja stanja pomoću senzora, odnosno analognih pinova, mogu se implementirati u realizirani sustav udaljene kontrole realiziran u ovom radu. Može se, primjerice, kontrolirati uređaj za dovod ili odvod tekućine, a

povratne informacije, odnosno stanje, se može čitati preko analognih pinova uz pomoć senzora razine tekućine. Nadalje, možemo npr. kontrolirati grijalicu, a uključujemo je po potrebi, ovisno o stanju temperature koje mjeri senzor temperature.

Bluetooth tehnologija idealno je i jeftino rješenje za kontrolu i nadzor uređaja unutar manjih prostora, primjerice u kući ili manjoj zgradi, dok upravljanje putem web stranice omogućuje globalnu kontrolu.

Slika 8-1 Relej modul

9. Zaključak

U ovom radu razmotrile su se mogućnosti udaljene kontrole i nadzora uređaja, a temelj svega je *Arduino* platforma. Njezina izrazita popularnost te otvorenost koda, uvelike su pomogli kod izrade samog rada. Veoma je bitno spomenuti modularnost same platforme koja joj pruža dodatne i brojne mogućnosti. Modularnost je i ovdje došla do izražaja jer se koristilo nekoliko modula koji *Arduino* platformi pružaju dodatne mogućnosti. U suštini, razvojem mobilne aplikacije (putem *App Inventora* i *Android Studia*), te korištenjem *Bluetooth* modula, omogućena je kontrola i nadzor uređaja idealna za manje udaljenosti. Isto tako, korištenjem *Ethernet* modula, realizirana je mogućnost globalnog upravljanja i nadzora uređaja. Ono bitno je realizacija mogućnosti udaljene kontrole samog *Arduina*, odnosno njegovih izlaza (kojima se nadalje mogu kontrolirati razni uređaji) i mogućnost dobivanja povratnih informacija sa njega samog. Dakle, obuhvaćene su sve navedene komponente, koristeći potrebnu programsku logiku te je iznesena početna ideja ovog rada.

U Varaždinu, _____ Potpis _____

10. Literatura

[1] https://www.arduino.cc/en/Main/Software, dostupno 10.07.2015.

[2] M. Kalapurić, R. Đurčević, R.: Bluetooth komunikacija, Fakultet prirodoslovno-matematičkih odgojnih znanosti, Sveučilište u Mostaru, 2012. https://www.scribd.com/doc/95849641/BLUETOOTH-TEHNOLOGIJA#scribd, dostupno 13.07.2015.

[3] M. Radovan, M.: Računalne mreže 1, Digital point tiskara, Rijeka, 2010.

[4] http://appinventor.mit.edu/explore/, dostupno 11.07.2015.

[5] M. Gargenta: Naučite Android, Dobar Plan, Zagreb, 2013.

[6]http://developer.Android.com/intl/zh-cn/guide/topics/connectivity/bluetooth.html, dostupno 15.07.2015

[7] D. Kermek: Izgradnja Web aplikacija - FOI, Fakultet organizacije i informatike, Varaždin, 2014.

[8] http://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf, dostupno 22.07.2015.

[9]http://portforward.com/english/routers/port_forwarding/ZTE/ZXV10-201L/radmin Remote _Administrator .htm-, dostupno 27.7.2015

[10] M. Mikac: Računalne mreže - RM, Odjel elektrotehnike, UNIN 2014.

[11] http://www.noip.com, dostupno 28.07.2015.

[12] http://www.biz.nf/, dostupno 05.08.2015.

[13] http://arduinokontrolainadzor.co.nf, dostupno 15.08.2015.

[14] https://www.arduino.cc/en/Reference/HomePage, dostupno 10.07.2015.

[15] https://www.arduino.cc/en/Reference/Ethernet, dostupno 20.07.2015.

[16]https://startingelectronics.org/tutorials/arduino/ethernet-shield-web-server-tutorial/web-server-LED-control/, dostupno 25.7.2015

[18] C. Schmitt: CSS Cookbook 2nd edition, O'Reilly Media, 2008

[19]http://www.tweaking4all.com/hardware/arduino/arduino-ethernet-data-push/, dostupno 11.08.2015

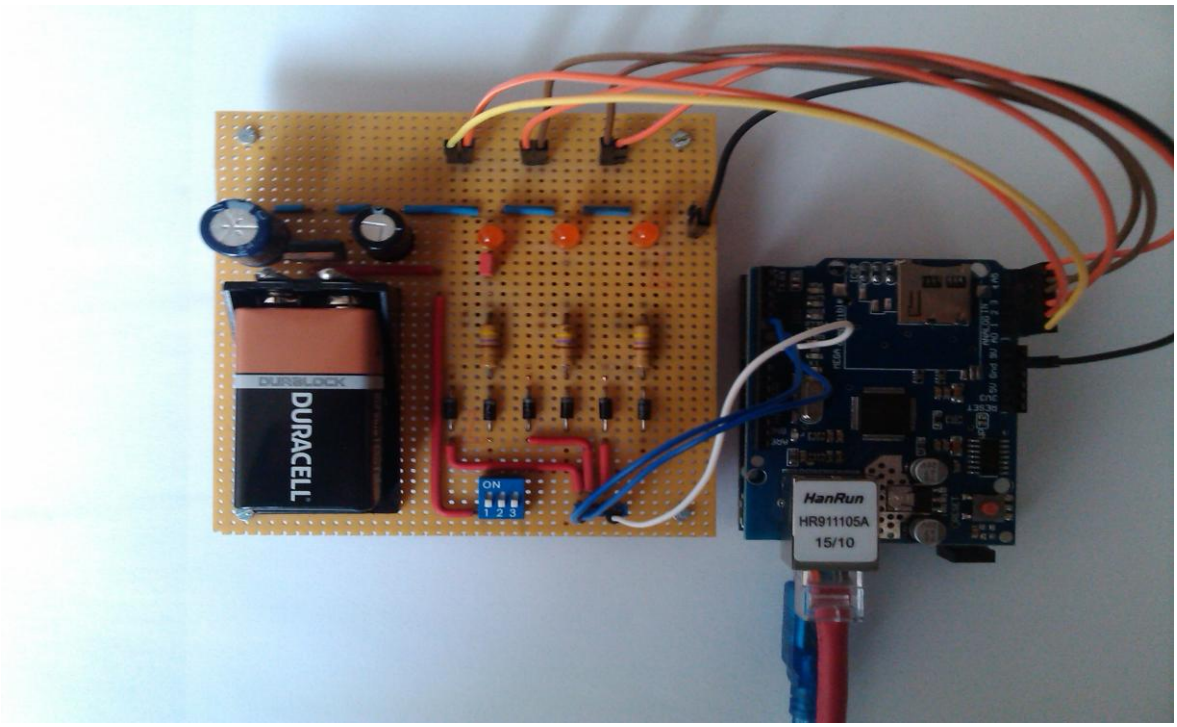
11. Popis slika i tabela

SLIKA 2-1. ARDUINO UNO R3 MODEL I NAJBITNIJI DIJELOVI 2
SLIKA 2-2. PRIKAZ IDE KORISNIČKOG SUČELJA 3
SLIKA 3-1. ELEKTRIČNA SHEMA IMPLEMENTIRANOG UREĐAJA 7
SLIKA 3-2. MJERENJE NAPONA BEZ OTPORNIKA OD 100 KΩ 7
SLIKA 3-3. MJERENJE NAPONA S OTPORNICIMA OD 100 KΩ 8
SLIKA 4-1. HC-06 BLUETOOTH MODUL 11
SLIKA 4-2. NAPONSKI DJELITELJ 12
SLIKA 4-3. UPARIVANJE SA HC-06 MODULOM 12
SLIKA 4-4. PROTOKOL KOMUNIKACIJE 13
SLIKA 4-5. PODACI KOJE GENERIRA FUNKCIJA CITAJSTANJE 19
SLIKA 4-6. APP INVENTOR DESIGNER DIO 20
SLIKA 4-7. APP INVENTOR BLOCK DIO 21
SLIKA 4-8. POČETNI ZASLON APLIKACIJE 22
SLIKA 4-9. OBAVIJEST ZA UKLJUČENJE BLUETOOTH-A 23
SLIKA 4-10. PROCEDURA KOJA UKLJUČUJE BLUETOOTH 23
SLIKA 4-11. LISTA UPARENIH BLUETOOTH UREĐAJA 23
SLIKA 4-12. USPOSTAVLJANJE BLUETOOTH VEZE 24
SLIKA 4-13. GUMBI KOJIMA ŠALJEMO PODATKE ARDUINU 24
SLIKA 4-14. GUMB KOJIM PREKIDAMO VEZU 25
SLIKA 4-15. PRIMANJE PODATAKA I PRIKAZ STANJA 25
SLIKA 4-16. IZGLED APLIKACIJE U APP INVENTORU 26
SLIKA 4-17. PRIKAZ DIJELOVA ANDROID STUDIA 27
SLIKA 4-18. ŽIVOTNI CIKLUS AKTIVNOSTI 29
SLIKA 4-19. PRIMJER XML KODA ZA GUMB 30
SLIKA 4-20. PRIMJER UKLJUČIVANJA BLUETOOTHA 31
SLIKA 4-21. NAJBITNIJI DIJELOVI DRUGE AKTIVNOSTI 32
SLIKA 4-22. IZGLED APLIKACIJA U ANDROID STUDIU 35
SLIKA 5-1. ETHERNET SHIELD SPOJEN NA ARDUINO 37
SLIKA 5-2. MODEL KLIJENT - POSLUŽITELJ 38
SLIKA 5-3. PRINCIP RADA ARDUINO POSLUŽITELJA 39
SLIKA 5-4. WEB STRANICA NA ARDUINU 40
SLIKA 5-5. OTVARANJA PORTA I POVEZIVANJE SA ADRESOM POSLUŽITELJA 42
SLIKA 5-6. KREIRANA MAPA - ARDUINOKONTROLAINADZOR.CO.NF 43
SLIKA 5-7. HTTP ODGOVOR POSLUŽITELJA 46
SLIKA 5-8. WEB STRANICA NA ZASEBNOM SERVERU 49
SLIKA 6-1. KOLIČINA GENERIRANIH PODATAKA KOD NE OPTIMIZIRANE VERZIJE 52
SLIKA 6-2. GENERIRANI PROMET U INTERVALIMA OD 10 SEKUNDI (OSVJEŽAVANJE SVAKE 3 SEKUNDE) 52
SLIKA 6-3. POTROŠNJA PROMETA BEZ AUTOMATSKOG OSVJEŽAVANJA STRANICE 53
SLIKA 6-4. GENERIRANI PROMET U INTERVALIMA OD 10 SEKUNDI (NA ZAHTJEV KORISNIKA) 53
SLIKA 8-1 RELEJ MODUL 57

TABLICA 6-1. POTROŠNJA PROMETA PRIJE I POSLIJE OPTIMIZACIJE 54
TABLICA 7-1. USPOREDBA CIJENA UREĐAJA 55
TABLICA 7-2. TROŠKOVI KUPNJE UREĐAJA 55

12. Prilozi

Prilog 1. Implementirani uređaj spojen na *Arduino* platformu



Prilog 2. *Manifest* datoteka izrađene aplikacije u *Android Studiu*

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.jogrobenski.arduinokontrolainadzor" >
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".BluetoothActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".kontrola"
            android:label="@string/title_activity_kontrola"
            android:configChanges="orientation"
            android:screenOrientation="portrait">
        </activity>
    </application>
</manifest>
```

Prilog 3. Metoda (*provjeri_stanje_bt*) kojom uključujemo *Bluetooth* na mobilnom uređaju

```
//metoda provjeri_stanje_bt koja uključuje Bluetooth na mobilnom uređaju
void provjeri_stanje_bt()
{
    //BluetoothAdapter predstavlja Bluetooth karticu mobilnog uređaja
    bluetooth = BluetoothAdapter.getDefaultAdapter();
    // ukoliko uređaj ne podržava Bluetooth,
    //ispiši poruku i izlazi iz aplikacije
    if(bluetooth==null) {
        Toast.makeText(getBaseContext(),
        "Vaš uređaj ne podržava  Bluetooth tehnologiju",
        Toast.LENGTH_SHORT).show();
        finish();// izlazi iz aplikacije
     }

     //ukoliko uređaj podržava Bluetooth, a nije uključen, uključi ga
     else if (!bluetooth.isEnabled()) {
        Intent ukljucibt = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(ukljucibt, 1);
    }
}
```

Prilog 4. Metoda *onResume* aktivnosti *kontrola*

```
public BluetoothSocket createBluetoothSocket(BluetoothDevice device) throws
IOException {
    //UUID (univerzalni identifikator Bluetooth servisa- 128 bitova)
    //za SPP uređaje
    return device.createRfcommSocketToServiceRecord(btmoduluuid);
}

@Override
//metoda onResume
public void onResume()
{
     super.onResume();
     Intent dohvati = getIntent();

     //dohvati MAC adresu iz prethodne aktivnosti  pomoću getStringExtra
     address = dohvati.getStringExtra(BluetoothActivity.adrese)
     BluetoothDevice device = bluetooth.getRemoteDevice(address);

     // pokušava kreirati Socket
     try {
         Socket = createBluetoothSocket(device);
     }
      //ukoliko ne uspije kreiranje Socketa, baci iznimku
      //(IOException),odnosno grešku vezanu za neuspjele ulazno
     //izlazne operacije - slanje, čitanje podataka i sl.)
     catch(IOException e){
         Toast.makeText(getBaseContext(),
         "Kreiranje Socketa nije uspjelo", Toast.LENGTH_SHORT).show();
     }

     //pokušava se povezati na HC-06 modul
     try {
         Socket.connect();
     }
     catch(IOException e){
         try {
              Socket.close();
         }
         catch(IOException e2){}
     }

      //kreiramo dretvu za komunikaciju i pokrećemo je
      mConnectedThread= new ConnectedThread(Socket);
      mConnectedThread.start();

      //Pomoću metode write() šaljemo  "t"→ test
      mConnectedThread.write("t");
}
```

Prilog 5. Klasa *ConnectedThread* (konstruktor dretve, metode run, write)

```
// klasa unutar koje kreiramo konstruktor dretve za komunikaciju, metodu run,
write, ulazno- izlazni tok...
private class ConnectedThread extends Thread
{
   // ulazni i izlazni tok
   private final InputStream ulaznitok;
   private final OutputStream izlaznitok;

   //konstruktor dretve(mConnectedThread)za komunikaciju
   public ConnectedThread(BluetoothSocket socket) {

      InputStream tmpIn = null;
      OutputStream tmpOut = null;
      try {
        tmpIn = socket.getInputStream();
        tmpOut = socket.getOutputStream();
      }
      catch(IOException e) { }

      ulaznitok= tmpIn;
      izlaznitok= tmpOut;
   }

   //run metoda u kojoj se podaci čitaju i prosljeđuju Handler-u
   public void run()
   {
       //za spremanje podataka
       byte[] buffer = new byte[1024];
       int bytes;
       //ukoliko postoje podaci za čitanje, čitaj ih
       while(true)
       {
         try {
            //podatke šaljemo Handler-u
            bytes = ulaznitok.read(buffer);
            String readMessage = new String(buffer, 0, bytes);
            handler.obtainMessage(handlerstanje,
            bytes, 1,readMessage).sendToTarget();
         }
         catch(IOException e) {
         break;
         }
      }
   }

   //write metoda - nju  pozivamo kod pisanja ,
   //odnosno slanja podataka. Njome   šaljemo a,b,c,d,e,f,t.
   public void write(String input) {

       byte[] msgBuffer = input.getBytes();

        try {
           izlaznitok.write(msgBuffer);
        }
        catch(IOException e) {
           //ukoliko kod povezivanja ne uspije pisati(poslati znak t ),
           //ne prebacuje nas na aktivnost kontrola već nas opet prebaci
           //na BluetoothActivity,odnosno početni zaslon
           Toast.makeText(getBaseContext(),"Povezivanje nije uspjelo,
           pokušajte ponovno", Toast.LENGTH_SHORT).show();
           finish(); // zatvara aktivnost
         }
    }
}
```

Prilog 6. HTTP zahtjev klijenta

```
GET / HTTP/1.1
Host: 192.168.5.177
Connection: keep-alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.134 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: hr-HR,hr;q=0.8,en-US;q=0.6,en;q=0.4
```

Prilog 7. Programski kod na *Arduinu* kad je poslužitelj (1. dio)

```
#include<SPI.h>
#include<Ethernet.h>
#define D5 5 //oznake za digitalne pinove od 5-7 (izlaz1,izlaz2 i izlaz3)
#define D6 6
#define D7 7
// MAC adresa Ethernet modula
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192,168,5,177); // IP adresa

EthernetServer server(80);  // server port
String HTTP_req; //za spremanje HTTP zahtjeva

void setup()
{
   Ethernet.begin(mac, ip);  // inicijalizacija Ethernet modula
   server.begin();           // pokretanje slušanja za klijentima
   Serial.begin(9600);       // korisno za praćenje podataka (zahtjeva)
   pinMode(D5, OUTPUT);      // postavljanje izlaznih pinova(5,6,7);
   pinMode(D6, OUTPUT);
   pinMode(D7, OUTPUT);
}

voidloop()
{
 EthernetClient client = server.available();  // pokušaj „dohvatiti" klijenta

   if (client) // provjera da li je klijent dohvaćen
   {
      boolean tekucalinijaprazna = true;
       while (client.connected())//ako je klijent uspješno spojen
       {
          if (client.available())// klijent ima podatke za poslati
          {
            char c = client.read(); // čita 1 byte (1 znak) od klijenta

            HTTP_req += c;  // sprema HTTP request po 1 znak

             // provjera da li je cijeli podatak primljen od klijenta
            if (c == '\n' && tekucalinijaprazna)
            {
                  // postavljanje LED dioda ovisno o GET parametru
                  postaviNovoStanje();

                  // šalji standardno zaglavlje HTTP odgovora
                  client.println("HTTP/1.1 200 OK");
                  client.println("Content-Type: text/html");
                  client.println("Connection: close");
                  client.println();

                  //početal web  stranice
                  client.println("<!DOCTYPE html>");
                  client.println("<html>");
                  client.println("<head>");
                  client.println("<meta http-equiv=\"refresh\"");
                  client.println("content=\"3\">");
                  client.println("<meta charset=\"UTF-8\">");
                  client.println("<meta name=\"viewport\" content");
                  client.println("=\"width=device-width, initial-scale=1\">");
                  client.println("<style>body{text-align:center;");
                  client.println(" background-color:#A0A0A0 ;}");
                  client.println(".c{color:#ff0000;} .z{color:#00ff00;}");
                  client.println("</style>");
                  client.println("</head>");
                  client.println("<bodystyle=\"text-align:center\">");
                  client.println("<h1>Arduino kontrola i nadzor</h1>");
                  client.println("<hr style=\"margin:10px 30%\"/>");
```

Programski kod na *Arduinu* kad je poslužitelj (2. dio)

```
                    //unos upravljačkih naredbi (a,b,c,d,e,f)
                    client.println("<form action=\"\" method=\"get\">");
                    client.println("<label for=\"LED\"><b>
                    client.println("Upravljanje izlazima(naredbe:");
                    client.println("[a,b],[c,d],[e,f]): </b></label>");
                    client.println("<input type=\"text\" name=\"LED\" value=\"\">");
                    client.println("<input type=\"submit\" value=\"primjeni\">");
                    client.println("</form>");
                    client.println("<hr style=\"margin:10px 30%\"/>");

                    // ispis stanja izlaza
                    for(int i=1; i<4; i++) citanjeIzlaz(client, i);

                    client.println("<hrstyle=\"margin:auto 30%\"/>");

                    // ispis stanja LED dioda
                    for(int i=1; i<4; i++) citanjeStanja(client, i);

                    client.println("<hrstyle=\"margin:auto 30%\"/>");
                    client.println("<div><p><b>JožeGrobenski</b>");
                    client.println("</p><p><b>SveučilišteSjever,");
                                    client.println("Varaždin</b></p></div>");
                    client.println("</body>");
                    client.println("</html>");// kraj web stranice
                    Serial.print(HTTP_req);
                    HTTP_req = "";
                    break;
                  }
                 // provjerava da li je konačno primljen i zadnji
                 //znak od klijenta (podatak uvijek završava sa \n)
                 //postavi da je linija prazna (gore potrebno kod
                 //provjere da li je klijent sve svoje podatke
                 //poslao odnosno server sve podatke primio)
                if (c == '\n') { tekucalinijaprazna = true; }

                 //inače postoji znak koji je server primio i oznaka
                 // za kraj prijenosa podataka se postavlja u false
                elseif (c != '\r'){ tekucalinijaprazna = false; }

            }
         }

    // davanje web  pregledniku vremena
    // za prihvat i prikaz podataka
    delay(1);
    client.stop();
   }
}
// funkcija služi za postavljanje statusa LED-ice, odnosno uključenje ili
//isključenje izlaza
void postaviNovoStanje()
{
    //čitanje trenutnog statusa LED1,odnosno izlaza1
    int LED1_status = digitalRead(D5);
    //čitanje trenutnog statusa LED2, odnonso izlaza2
    int LED2_status = digitalRead(D6);
    //čitanje trenutnog statusa LED3, odnosno izlaza3
    int LED3_status = digitalRead(D7);

    //provjera da li je došao zahtjev za uključivanjem LED1
    //(parametar LED sa vrijednosti a)
    int x = HTTP_req.indexOf("LED=a");
    //dio sa x<10 označava da je pronađen "LED=a" unutar
    // prvih 10 znakova GET metode
    if (x>-1 && x<10) LED1_status = 1;
```

Programski kod na Arduinu kad je poslužitelj (3. dio)

```
      //provjera da li je došao zahtjev za isključivanjem
      //LED 1 (parametar LED sa vrijednosti b)
      x = HTTP_req.indexOf("LED=b");
      if (x>-1 && x<10) LED1_status = 0;

      //provjera da li je došao zahtjev za uključivanjem
      // LED2 (parametar LED sa vrijednosti c)
      x = HTTP_req.indexOf("LED=c");
      if (x>-1 && x<10) LED2_status = 1;

      //provjera da li je došao zahtjev za isključivanjem
      //LED2 (parametar LED sa vrijednosti d)
      x = HTTP_req.indexOf("LED=d");
      if (x>-1 && x<10) LED2_status = 0;

      //provjera da li je došao zahtjev za uključivanjem
      LED3 (parametar LED sa vrijednosti e)
      x = HTTP_req.indexOf("LED=e");
      if (x>-1 && x<10) LED3_status = 1;

      //provjera da li je došao zahtjev za isključivanjem
      LED3 (parametar LED sa vrijednosti f)
      x = HTTP_req.indexOf("LED=f");
      if (x>-1 && x<10) LED3_status = 0;

      // preko izlaznih pinova postavljamo LED1 na uključeno
      //ili isključeno stanje- ovisno o prethodnom stanju i GET parametru
      // isto vrijedi i za LED2, te LED3
      digitalWrite(D5, LED1_status);
      digitalWrite(D6, LED2_status);
      digitalWrite(D7, LED3_status);
}

// funkcija služi za čitanje stanja LED dioda (ovisno o "id" parametru -
provjerava se LED1, LED2 ili LED3)
void citanjeStanja(EthernetClient c, int id)
{
      // ovo su pinovi koji se provjeravaju
      //(ovisno o kojoj je LED diodi riječ tako se
      //postavljaju unutar sljedećeg "if-else"
      int pin1, pin2;
      // ako je "id" parametar = 1 onda se radi o LED1
      if(id==1)      { pin1 = A0; pin2 = A1; }
      // ako je "id" parametar = 2 onda se radi o LED2
      elseif(id==2) { pin1 = A2; pin2 = A3; }
      // inače se radi o LED3
      else           { pin1 = A4; pin2 = A5; }

      bool a; //oznaka da li je LED dioda uključena ili isključena
      float v  = analogRead(pin1)* (5.0 / 1023.0); //pretvorba
      float m  = analogRead(pin2) * (5.0 / 1023.0); //pretvorba
      float r = v - m; // računanje razlike napona
      // određivanje stanja
      if (r > 0.1)  a = true; // uključeno
      else          a = false; // isključeno

      // određujemo o kojoj LED diodi se radi
      c.print("<p>");
      if(id==1)      c.print("<b>LED 1</b>");
      else if(id==2) c.print("<b>LED 2</b>");
      else           c.print("<b>LED 3</b>");

      ispisujemo pomoću HTML koda stanja
      if(a) c.println(": <span class=\"z\">uključeno</span></p>");
     else  c.println(": <span class=\"c\">isključeno</span></p>");
}
```

Programski kod na Arduinu kad je poslužitelj (4. dio)

```
// funkcija za čitanje stanja izlaza
void citanjeIzlaz(EthernetClient c, int id)
{
      // pin koji služi za spremanje D5, D6 ili D7 ovisno o parametru
      int pin;
      if(id==1)      pin = D5; //određivanje izlaza
      else if(id==2) pin = D6;
      else           pin = D7;

      c.print("<p>"); //ispis
      if(id==1)        c.print("<b>IZLAZ 1</b>");
      else if(id==2)   c.print("<b>IZLAZ 2</b>");
      else             c.print("<b>IZLAZ 3</b>");

      if(digitalRead(pin))c.println(":<spanclass=\"z\">uključeno</span></p>");
      else c.println(": <span class=\"c\">isključeno</span></p>");
}
```

Prilog 8. Programski kod kad je *Arduino* klijent (1. dio)

```
#include <Ethernet.h>
#include <SPI.h>

//MAC adresa Ethernet modula
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
EthernetClient klijent;

// server na kojeg se spajamo,šaljemo zahtjeve i čitamo podatke
char server[] = "arduinokontrolainadzor.co.nf";

//izlazni pinovi Arduina
int izlaz1 = 5;
int izlaz2 = 6;
int izlaz3 = 7;
//varijable u koje spremamo stanja izlaza Arduina
int stizl1;
int stizl2;
int stizl3;
// varijable za mjerenje napona
float razlika1;
float razlika2;
float razlika3;
//varijable u koje spremamo stanja LED dioda
int stled1;
int stled2;
int stled3;

//pretvorba
float pretvorba = (5.0/1023);

voidsetup()
{
   Serial.begin(9600);
   pinMode (izlaz1, OUTPUT);
   pinMode (izlaz2, OUTPUT);
   pinMode (izlaz3, OUTPUT);
   // ovom naredbnom zapravo "aktiviramo" DHCP
   //i automatski nam se
   // dodjeljuju mrežni parametri ( P adresa)
   Ethernet.begin(mac);
   Serial.print("IP adresa uredaja:  ");
   //ispisuje dodijeljenju IP adresu Ethernet modula
   Serial.println(Ethernet.localIP());

}

voidloop()
{
   //stanje izlaz 1
   if (digitalRead(izlaz1)) stizl1 = 1; //uključen
   else                     stizl1 = 0; //isključen

   //stanje izlaz 2
   if (digitalRead(izlaz2)) stizl2 = 1; //uključen
   else                     stizl2 = 0; //isključen

   //stanje izlaz 3
   if (digitalRead(izlaz3)) stizl3 = 1; //uključen
   else                     stizl3 = 0; //isključen
```

Programski kod kad je *Arduino* klijent (2. dio)

```
//stanje LED1
float veci1  = analogRead(A0);
float manji1 = analogRead(A1);
razlika1 = (veci1 - manji1)* pretvorba;
if (razlika1 > 0.1)  stled1 = 1; //uključena
else                 stled1 = 0; //isključena

//stanje LED2
float veci2  = analogRead(A2);
float manji2 = analogRead(A3);
razlika2 = (veci2 - manji2)* pretvorba;
if (razlika2 > 0.1)  stled2 = 1; //uključena
else                 stled2 = 0; //isključena

//stanje LED3
float veci3  = analogRead(A4);
float manji3 = analogRead(A5);
razlika3 = (veci3 - manji3) * pretvorba;
if (razlika3 > 0.1) stled3 = 1; //uključena
else                stled3 = 0; //isključena

//omogućava obnovu mrežnih
//parametara (IP adrese) pomoću DHCP protokola
Ethernet.maintain();

//spajanje na server i slanje stanja izlaza i led dioda
if (klijent.connect(server, 80))
{
  Serial.println("slanje stanja");
  //slanje podataka GET metodom u datoteku dolaznipodaci.php
  klijent.print("GET http://arduinokontrolainadzor.co.nf/dolaznipodaci.php?");
  klijent.print("izl1=");
  klijent.print(stizl1);
  klijent.print("&izl2=");
  klijent.print(stizl2);
  klijent.print("&izl3=");
  klijent.print(stizl3);
  klijent.print("&led1=");
  klijent.print(stled1);
  klijent.print("&led2=");
  klijent.print(stled2);
  klijent.print("&led3=");
  klijent.print(stled3);
  klijent.println(" HTTP/1.1");
  klijent.println("Host: arduinokontrolainadzor.co.nf" );
  klijent.println("Connection: close" );
  klijent.println();
  klijent.stop();
  delay(300);
}

else  { Serial.println("podaci nisu poslani");  }
```

Programski kod kad je *Arduino* klijent (3. dio)

```
    //čitanje podataka za izlaz 1
    if (klijent.connect(server, 80))
    {
     Serial.println("povezano 1");
     //zahtjev za datoteku izlaz1.txt
     klijent.println("GET http://arduinokontrolainadzor.co.nf/izlaz1.txt HTTP/1.1");
     klijent.println("Host: arduinokontrolainadzor.co.nf");
     klijent.println("Connection: close");
     //kraj zahtjeva- prazna linija
     klijent.println();

     //klijent se smatra povezanim, unatoč tome što
     // je veza zatvorena (Connection:close),
     //ukoliko ima još podataka za čitanje
     while (klijent.connected())
     {
       // vraća broj bajtova dostupan za čitanje (od poslužitelja)
       while (klijent.available())
       {
         // ukoliko ima podataka za čitanje, čitaj znak po znak
         // i spremaj u varijablu ch1
         char ch1 = klijent.read();
         //Serial.print(ch1);

         // kad je ch1 jednako znaku *, uključi izlaz 1
         if (ch1 == '*') {
           digitalWrite(izlaz1, HIGH);
           Serial.println("izlaz 1 ukljucen");
         }

         // ako je ch1 jednako znaku %, isključi izlaz 1
         else if (ch1 == '%'){
           digitalWrite(izlaz1, LOW);
           Serial.println("izlaz 1 iskljucen");
         }

       }
     }
     //odspajanje
     klijent.stop();
     delay(300);
    }

    else {  Serial.println("citanje izlaz1-- citanje nije uspjelo"); }
```

Programski kod kad je *Arduino* klijent (4. dio)

```
    // čitanje podataka za izlaz 2
    if (klijent.connect(server, 80)){
       Serial.println ("povezano 2");
       // zahtjev za izlaz2.txt
       klijent.println("GET http://arduinokontrolainadzor.co.nf/izlaz2.txtHTTP/1.1");
       klijent.println("Host: arduinokontrolainadzor.co.nf");
       klijent.println("Connection: close");
       klijent.println();

       while (klijent.connected())
       {
         while (klijent.available())
         {
           char ch2 = klijent.read();

            if (ch2 == '>'){ //uključi izlaz2
              digitalWrite(izlaz2, HIGH);
              Serial.println("izlaz 2 ukljucen");
            }
            else if (ch2 == '<'){ //isključi izlaz2
              digitalWrite(izlaz2, LOW);
              Serial.println("izlaz 2 iskljucen");
            }
         }
       }
      klijent.stop();
      delay(300);
    }

    else { Serial.println("citanje izlaz 2-- spajanje nije uspjelo"); }

    // čitanje podata za izlaz 3
    if (klijent.connect(server, 80)) {

       Serial.println ("povezano 3");
       klijent.println("GET http://arduinokontrolainadzor.co.nf/izlaz3.txtHTTP/1.1");
       klijent.println("Host: arduinokontrolainadzor.co.nf");
       klijent.println("Connection: close");
       klijent.println();

       while (klijent.connected())
       {
         while (klijent.available())
         {
           char ch3 = klijent.read();
           // Serial.print(ch3);

           if (ch3 == '^'){
             digitalWrite(izlaz3, HIGH);
             Serial.println("izlaz 3 ukljucen");C
           }
           else if (ch3 == '!'){
             digitalWrite(izlaz3, LOW);
             Serial.println("izlaz 3 iskljucen");
           }
         }
       }

       klijent.stop();
       delay(300);
    }
    else { Serial.println("citanje izlaz 3-- spajanje nije uspjelo"); }

}// zatvara loop
```

Prilog 9. Programski kod web stranice *index.php* (1. dio)

```
<!DOCTYPE html>
<html>
<head>
   // za znakove č,ć,š i slične
   <meta  charset="UTF-8">
   //automatsko osvježavanje stranice
   <meta http-equiv="refresh" content="3" >
   <title>Arduino kontrola i nadzor</title>
   // početak CSS-a
    <style>
    html,
    body {
      margin:0;
      padding:0;
      height:100%;
      font-family:"Arial Black", "Arial Black", Gadget, sans-serif;
      background-color:#D8D8D8  ;
    }
    #omotac {
      min-height:100%;
      position:relative;
    }
    #zaglavlje{
      background:#A8A8A8;
      height: 60px;
      padding:10px;
    }
    #sadrzaj{
      padding-bottom:30px;
    }
    #podnozje{
      background:#A8A8A8;
      width:100%;
      height:30px;
      position:absolute;
      bottom:0;
      left:0;
    }
    h1{
      margin:auto;
      text-align:center;
      padding:8px;
      color:#002447;
    }
    h3{
      text-align:center;
      color:#002447;
      margin: 6px;
      font-size:16 px;
      padding:2px;
    }
```

Programski kod web stranice *index.php* (2. dio)

```
    button{
      border-top: 1px solid #f0f9ff;
      background: #131b21;
      background: -webkit-gradient(linear, left top, leftbottom,
      from(#2e3c45),     to(#131b21));
      background: -webkit-linear-gradient(top, #2e3c45, #131b21);
      background: -moz-linear-gradient(top, #2e3c45, #131b21);
      background: -ms-linear-gradient(top, #2e3c45, #131b21);
      background: -o-linear-gradient(top, #2e3c45, #131b21);
      padding: 10.5px 21px;
      -webkit-border-radius: 9px;
      -moz-border-radius: 9px;
      border-radius: 9px;
      margin-left: 3px;
      margin-right: 3px;
      margin-top:5px;
      margin-bottom:5px;
      -webkit-box-shadow: rgba(0,0,0,1) 0 1px 0;
      -moz-box-shadow: rgba(0,0,0,1) 0 1px 0;
      box-shadow: rgba(0,0,0,1) 0 1px 0;
      text-shadow: rgba(0,0,0,.4) 0 1px 0;
      color: #ffffff;
      font-size: 13px;
      font-family: Georgia, Serif;
      text-decoration: none;
      vertical-align: middle;
    }
    button:hover{
      border-top-color: #212a30;
      background: #212a30;
      color: #ffffff;
    }
    .zeleni{
         width:1%;
         padding:10px 6px;
         margin:0 auto;
         border-radius:100%;
         background-color:green;
    }
    .crveni{
         width:1%;
         padding:10px 6px;
         margin:0 auto;
         border-radius:100%;
         background-color:red;
    }
    .infotekst{
         color: #002447;
         text-align:center;
         margin-top:5px;
         font-size:22 px;
    }
    </style> // kraj CSS-a
</head>
```

Programski kod web stranice *index.php* (3. dio)

```
//dio web stranice koji je vidljiv korisniku
<body>
 <div id="omotac">
  <div id="zaglavlje">
  <h1>Arduino kontrola i nadzor</h1>
  </div>
  <div id="sadrzaj">
   <h3>Upravljanje izlazima</h3>

    //upravljanje izlazom 1
   <center><a href ="izlaz1.php?izlazprvi=*">
   <button>Uključi izlaz1</button></a>
   <ahref ="izlaz1.php?izlazprvi=%">
   <button>Isključi izlaz 1</button></a></center>

    //upravljanje izlazom 2
   <center><a href ="izlaz2.php?izlazdrugi=>">
   <button>Uključi izlaz 2</button></a>
   <a href ="izlaz2.php?izlazdrugi=<">
   <button>Isključi izlaz2</button></a></center>

    // upravljanje izlazom 3
    <center><a href ="izlaz3.php?izlaztreci=^">
    <button>Uključi izlaz 3</button></a>
    <a href ="izlaz3.php?izlaztreci=!">
    <button>Isključi izlaz3</button></a></center>
    <hrwidth="50%"/>

<?php // početak php skripte

       //prikaz stanja izlaz 1
       echo'<h3>Izlaz 1 stanje</h3>';
       $stizl1 = file_get_contents("stanjeizlaz1.txt");
       if ($stizl1=="1") {
          echo'<div class="zeleni"></div>';
       }elseif ($stizl1=="0") {
          echo '<div class="crveni"></div>';
       }

       //prikaz stanja izlaz 2
       echo '<h3>Izlaz 2 stanje</h3>';
       $stizl2 = file_get_contents("stanjeizlaz2.txt");
       if ($stizl2=="1") {
          echo'<div class="zeleni"></div>';
       }elseif($stizl2=="0"){
          echo '<div class="crveni"></div>';
       }

       //prikaz stanja izlaz 3
       echo'<h3>Izlaz 3 stanje</h3>';
       $stizl3 = file_get_contents("stanjeizlaz3.txt");
       if($stizl3=="1"){
          echo'<div class="zeleni"></div>';
       }elseif($stizl3=="0"){
          echo '<div class="crveni"></div>';
       }
       echo'<hrwidth="50%"/>';
```

Programski kod web stranice *index.php* (4. dio)

```
        // prikaz stanja LED 1
        echo'<h3>LED 1 stanje</h3>';
        $stled1 = file_get_contents("stanjeled1.txt");
        if($stled1=="1"){
           echo'<div class="zeleni"></div>';
        }elseif($stled1=="0"){
           echo '<div class="crveni"></div>';
        }

        //prikaz stanja LED 2
        echo'<h3>LED 2 stanje</h3>';
        $stled2 = file_get_contents("stanjeled2.txt");
        if($stled2 == "1"){
           echo'<div class="zeleni"></div>';
        }elseif($stled2=="0"){
           echo '<div class="crveni"></div>';
        }

        //prikaz stanja LED 3
        echo'<h3>LED 3 stanje</h3>';
        $stled3 = file_get_contents("stanjeled3.txt");
        if($stled3=="1"){
           echo'<div class="zeleni"></div>';
        }elseif($stled3=="0"){
           echo '<div class="crveni"></div>';
        }
        echo'<hrwidth="50%"/>';

?> // završetak php skripte

</div> // zatvara sadrzaj

 <div id="podnozje">
 <div class="infotekst">Izradio: Jože Grobenski,
 Sveučiliste Sjever, Varaždin </div>
 </div> // zatvara podnozje

</div> // zatvara omotac
</body>
</html> //kraj HTML dokumenta
```

Prilog 10. Optimizirana funkcija *citajstanje*

```
void citajstanje()
{
  // za spremanje trenutnog stanja, deklarirana na
  // početku programa, tipa String
  novoStanje = "";

  novoStanje += "*"; //označava početak podataka

  //čitanje stanja izlaza1
  if (digitalRead(izlaz1)) novoStanje += "prvion"; //uključen
  else                     novoStanje += "prviof"; //isključen

  // služi za odvajanje podataka (stanja izlaza i LED dioda)
  novoStanje += ":";

  //čitanje stanja za izlaza2
  if (digitalRead(izlaz2)) novoStanje += "drugion"; //uključen
  else                     novoStanje += "drugiof"; //isključen
  novoStanje += ":";//odvajanje podataka

  //čitanje stanja za izlaz3
  if (digitalRead(izlaz3)) novoStanje += "trecion";  //uključen
  else                     novoStanje += "treciof";  //isključen
  novoStanje += ":"; //odvajanje podataka

  //čitanje stanja LED1
  float veci1  = analogRead(A0);
  float manji1 = analogRead(A1);
  //Pad napona na R1
  razlika1 = (veci1 - manji1) * pretvorba;
  //ako je pad napona veći od 0,1V LED-ice svijetle u suprotnom ne
  if (razlika1 > 0.1 ) novoStanje += "LED1on"; // uključena
  else                 novoStanje += "LED1of"; //isključena
  novoStanje += ":"; //odvajanje podataka

  //čitanje stanja LED2
  float veci2  = analogRead(A2);
  float manji2 = analogRead(A3);
  //Pad napona na R2
  razlika2 = (veci2 - manji2) * pretvorba;
  if (razlika2 > 0.1 ) novoStanje += "LED2on"; // uključena
  else                 novoStanje += "LED2of"; //isključena
  novoStanje += ":"; // odvajanje podataka

  //čitanje stanja LED3
  float veci3  =  analogRead(A4);
  float manji3 =  analogRead(A5);
  //Pad napona na R3
  razlika3 = (veci3 - manji3) * pretvorba;
  if (razlika3 > 0.1)  novoStanje += "LED3on"; //uključena
  else                 novoStanje += "LED3of"; //isključena

  novoStanje += ":";  //odvajanje podataka
  novoStanje += "#";  //označava kraj podataka

  //uspoređuju se  trenutno stanje sa prijašnjim stanjem
  if(novoStanje != staroStanje) {
    //ispisuje stanja (slika 4-5, poglavlaje 4.4)
    Serial.println(novoStanje);
    //varijabla staroStanje deklarirana je
    // na početku programa i tipa je String
    staroStanje = novoStanje;
  }
  delay(10);
}
```

Prilog 11. Optimizirani programski kod kad je *Arduino* klijent (1.dio)

```
#include <Ethernet.h>
#include <SPI.h>

//MAC adresa Ethernet modula
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
EthernetClient klijent;

// server na kojeg se spajamo,šaljemo zahtjeve i čitamo podatke
char server[] = "arduinokontrolainadzor.co.nf";

//izlazni pinovi Arduina
int izlaz1 = 5;
int izlaz2 = 6;
int izlaz3 = 7;

//pretvaranje u volte
float pretvorba = (5.0 / 1023.0);
//koristimo polje za spremanje stanja izlaza i LED dioda
//indeksi 0,1,2 = led1,led2,led3
//indeksi 3,4,5 = izlaz1,izlaz2,izlaz3
int stanje_staro[6];
int stanje_novo[6];

//pomoćna varijabla kod testiranja promjene stanja
bool obnovi;

void setup() {
  Serial.begin(9600);
  pinMode (izlaz1, OUTPUT);
  pinMode (izlaz2, OUTPUT);
  pinMode (izlaz3, OUTPUT);
  //ovom naredbnom DHCP nam automatski
  //dodjeljuje mrežne parametre (IP adresa)
  Ethernet.begin(mac);
  Serial.print("IP adresa uredaja: ");
  //ispisuje dodijeljenju IP adresu Ethernet modula
  Serial.println(Ethernet.localIP());
  //postavi početna stanja
  //na početku postavljamo različita da se pošalju
  for(int i=0; i<6; i++) stanje_staro[i] = -1;
  for(int i=0; i<6; i++) stanje_novo[i] = 0;
}
```

Optimizirani programski kod kad je *Arduino* klijent (2. dio)

```
void loop()
{
  //stanje izlaz1, izlaz2, izlaz3
  stanje_novo[3] = digitalRead(izlaz1);
  stanje_novo[4] = digitalRead(izlaz2);
  stanje_novo[5] = digitalRead(izlaz3);

  //stanje LED1
  float veci1   = analogRead(A0) * pretvorba;
  float manji1  = analogRead(A1) * pretvorba;
  stanje_novo[0] = ((veci1-manji1)>0.1 ? 1 : 0);

  //stanje LED2
  float veci2  = analogRead(A2) * pretvorba;
  float manji2 = analogRead(A3) * pretvorba;
  stanje_novo[1] = ((veci2-manji2)>0.1 ? 1 : 0);

  //stanje LED3
  float veci3  = analogRead(A4) * pretvorba;
  float manji3 = analogRead(A5) * pretvorba;
  stanje_novo[2] = ((veci3-manji3)>0.1 ? 1 : 0);

  //omogućava obnovu IP adrese pomoću DHCP protokola
  Ethernet.maintain();


  obnovi = false;
  //provjeri da li se stanje promijenilo
  for(int i=0; i<6; i++) {
    if(stanje_novo[i] != stanje_staro[i]) {
      obnovi = true;
      break;
    }
  }

  //ako je došlo do promjene, šalji stanja
  if(obnovi) {
    // spremi novo stanje kao staro
    for(int i=0; i<6; i++) stanje_staro[i] = stanje_novo[i];

          //slanje stanja izlaza i LED dioda
          if (klijent.connect(server, 80)) {
            Serial.println(" slanje stanja ");
            //voditi računa o print i println
            klijent.print("GET http://arduinokontrolainadzor.co.nf/postavi.php?stanje=");
             //ispisuje stanja
             for(int i=0; i<6; i++)   klijent.print(stanje_novo[i]);
            klijent.println(" HTTP/1.1");
            klijent.println("Host: arduinokontrolainadzor.co.nf" );
            klijent.println("Connection: close" );
            klijent.println();
            klijent.stop();
            delay(1000);
          }
  else {  Serial.println(" podaci nisu poslani");  }
  }
```

Optimizirani programski kod kad je *Arduino* klijent (3. dio)

```
  //čitaj podatke u dohvati.php i ovisno i njima, upravljaj izlazima
  if (klijent.connect(server, 80))  {

    klijent.println("GET http://arduinokontrolainadzor.co.nf/dohvati.php HTTP/1.1");
    klijent.println("Host: arduinokontrolainadzor.co.nf");
    klijent.println("Connection: close");
    klijent.println();
    Serial.println("citanje stanja");
    while (klijent.connected()){
      while (klijent.available()){
        char ch = klijent.read();
        switch(ch) {
          //ako pročita znak *, uključi izlaz1
          case '*':
            digitalWrite(izlaz1, HIGH);
            Serial.println("izlaz 1 ukljucen");
            break;
           //ako pročita znak %, isključi izlaz1, isti princip za ostale izlaze
          case '%':
            digitalWrite(izlaz1, LOW);
            Serial.println("izlaz 1 iskljucen");
            break;
          case '}':
            digitalWrite(izlaz2, HIGH);
            Serial.println("izlaz 2 ukljucen");
            break;
          case '{':
            digitalWrite(izlaz2, LOW);
            Serial.println("izlaz 2 iskljucen");
            break;
          case '^':
            digitalWrite(izlaz3, HIGH);
            Serial.println("izlaz 3 ukljucen");
            break;
          case '!':
            digitalWrite(izlaz3, LOW);
            Serial.println("izlaz 3 iskljucen");
            break;

          default: break;
        }
      }
    }
    klijent.stop();
    Serial.println();
    delay(1000);
  }
  else {  Serial.println(" podaci nisu procitani");  }

}//zatvara loop
```

Prilog 12. Izmjene na web stranici (*index.php*)

```
//na početku stranice umetnuta PHP skripta sa kojom se dohvaćaju stanja
<?php
//dohvati stanja koja je Arduino poslao
//na temelju njih prikazuj stanja na stranici
$stanje = file_get_contents("stanje.txt");
//spremaj u varijable (članove polja)
$stled1 = $stanje[0];
$stled2 = $stanje[1];
$stled3 = $stanje[2];
$stizl1 = $stanje[3];
$stizl2 = $stanje[4];
$stizl3 = $stanje[5];
?>


//izmjena upravljačkih naredbi i dodan gumb Dohvati stanja
<h3>Upravljanje izlazima</h3>

//upravljanje izlazom 1
<center><a href
="postavi.php?s0=1&id=izl1&stanje=1&akcija=pocetna"><button>Uključi izlaz
1</button></a>
<a href ="postavi.php?s0=1&id=izl1&stanje=0&akcija=pocetna"><button>Isključi
izlaz 1</button></a></center>

//upravljanje izlazom 2
<center><a href
="postavi.php?s0=1&id=izl2&stanje=1&akcija=pocetna"><button>Uključi izlaz
2</button></a>
<a href ="postavi.php?s0=1&id=izl2&stanje=0&akcija=pocetna"><button>Isključi
izlaz 2</button></a></center>

//upravljanje izlazom 3
<center><a href
="postavi.php?s0=1&id=izl3&stanje=1&akcija=pocetna"><button>Uključi izlaz
3</button></a>
<a href ="postavi.php?s0=1&id=izl3&stanje=0&akcija=pocetna"><button>Isključi
izlaz 3</button></a></center>
 <hr width="50%"/>

//gumb za dohvat stanja
<center><a href ="index.php"><button>Dohvati stanje</button></a></center>
<hr width="50%"/>
```

Prilog 13. PHP skripta *postavi.php*

```
<?php
/*
 * datoteka stanje:
 *   indeksi: 012345
 *   [stled1][stled2][stled3][stizl1][stizl2][stizl3]
 */
//preuzmi staro stanje
$stanje = file_get_contents("stanje.txt");
//odradi promjene
//dohvati sa index.php za izl1, izl2 i izl3
if(isset($_GET["id"]) && isset($_GET["stanje"]) && ($_GET["stanje"]==0 ||
$_GET["stanje"]==1) ) {
	switch($_GET["id"]) {
		case "led1": $stanje[0] = $_GET["stanje"]; break;
		case "led2": $stanje[1] = $_GET["stanje"]; break;
		case "led3": $stanje[2] = $_GET["stanje"]; break;
		case "izl1": $stanje[3] = $_GET["stanje"]; break;
		case "izl2": $stanje[4] = $_GET["stanje"]; break;
		case "izl3": $stanje[5] = $_GET["stanje"]; break;
		default: break;
	}
//dohvati stanja koja šalje Arduino
} else if(isset($_GET["stanje"]) ) {
	$stanje2 = $_GET["stanje"];
	$ok = false;
	if(strlen($stanje2) == 6) {
		$ok = true;
		for($i=0; $i<6; $i++) {
			if($stanje2[$i]!=0 && $stanje2[$i]!=1)
				$ok = false;
		}
	}
	if($ok) $stanje = $stanje2;
}

// spremi novo stanje-- na temelju tog stanja se
//prikazuju stanja na web stranici (njih šalje Arduino)
$datNaziv = "stanje.txt";

//ako se dobije zahtjev za upravljanje sa
//stranice(s0 postavljen i jednak jedinici)
//zapisuj u stanje0.txt vrijednosti(0 ili 1) koje se dohvaćaju u dohvati.php
// i na temelju njih ispisuju znakovi za upravljanje izlazima koje
// Arduino čita
if(isset($_GET["s0"]) && $_GET["s0"]==1) {
	$datNaziv = "stanje0.txt";
}

$dat = fopen($datNaziv, "w");
fwrite($dat, $stanje);
fclose($dat);

//vraća se na glavnu stranicu (index.php)
if(isset($_GET["akcija"])   &&   $_GET["akcija"]=="pocetna")   header("Location:
index.php");

?>
```

Prilog 14. PHP skripta *dohvati.php*

```
<?php
// preuzmi stanje za upravljanje
$stanje = file_get_contents("stanje0.txt");

// ovisno o vrijednosti u stanje0.txt, ispisuju se određeni znakovi
//upravljanje izlazom1
//ako je vrijednost člana polja sa indeksom 3 jednaka 1,
//ispiši * (uključi izlaz1) , za 0 ispiši % (isključi izlaz1)
echo ($stanje[3] ? "*" : "%");

//upravljanje izlazom 2
//za 1 ispiši } (uključi izlaz2), inače { (isključi izlaz2)
echo ($stanje[4] ? "}" : "{");

//upravljanje izlazom 3
//znak ^ za uključiti izlaz3, a znak ! za isključiti
echo ($stanje[5] ? "^" : "!");
?>
```