Izrada aplikacije za web stvari

Filip, Hrstić

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Filip Hrstić

IZRADA APLIKACIJE ZA WEB STVARI

DIPLOMSKI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Filip Hrstić

Matični broj: 44096/15–R

Studij: Baze podataka i baze znanja

IZRADA APLIKACIJE ZA WEB STVARI

DIPLOMSKI RAD

Mentor :

Doc. dr. sc. Darko Andročec

Varaždin, veljača 2022.

Filip Hrstić

Izjava o izvornosti

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

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema ovog diplomskog rada je razvoj aplikacije za web stvari. U prvom dijelu rada je detaljno je razrađena teorijska podloga vezana uz Internet stvari. Uključuje povijest interneta stvari, opis osnovnih karakteristika i arhitekture. Navedeni su najpoznatiji uređaji i senzori s naglaskom na one korištene u praktičnom dijelu rada. Pojam interneta stvari bilo je potrebno pobliže objasniti jer je usko vezan uz pojam weba stvari. Nakon podrobnog presjeka navedenih pojmova opisan je praktični dio. Sastoji se od programiranja IoT uređaja i ostvarivanja interakcije uređaja s Firebase platformom. Temelj praktičnog dijela je razvoj aplikacije korištenjem Flutter okvira. Aplikacija je realizirana kao centralna, potpuno modularna korisnička jedinica za daljinsko upravljanje IoT uređajima.

Ključne riječi: aplikacija; Arduino; Firebase; Flutter; iot; wot

Sadržaj

1. Uvod

Internet stvari je relativno nov koncept čija primjena se razvojem tehnologije eksponencijalno povećava. Koristi se gotovo u svim sferama života i danas je pristupačan kao nikada ranije. Upravo su se zbog nekontroliranog rasta interneta stvari izrodili problemi interoperabilnosti te nedostatak standardiziranih metoda i protokola. Rješenje tog problema pronađeno je u već postojećim i stabilnim web standardima zbog čega nastaje pojam weba stvari. Sve veća važnost interneta i weba stvari u današnjem vremenu primarna je motivacija za izradu ovog diplomskog rada. U nastavku rada nastojat ću objasniti sličnosti i razlike tih dvaju pojmova. Koristeći stečena znanja iz teorijskog dijela rada, implementirat ću jednostavan sustav pametnog doma koristeći pristupačne razvojne ploče i senzore. Poštujući arhitekturu weba stvari, omogućit ću uređajima komunikaciju s web servisima i postići daljinsko upravljanje uređajima putem mobilne aplikacije.

Flutter je sve popularniji Googleov okvir za razvoj višeplatformskih mobilnih aplikacija visokih performansi. Budući da nije bilo strogo definiranog jezika za razvoj aplikacije, odlučio sam se za ovaj moderan i intuitivan okvir. U konačnici, cilj rada je shvatiti pojam weba stvari te pomoću naučenih koncepata razviti pametni dom i mobilnu aplikaciju za upravljanje istim.

2. Metode i tehnike rada

Izrada ovog rada temelji se na metodama agilnog razvoja softvera stečenim na fakultetu. Agilni pristup prikladan je za dinamički razvoj softvera promjenjivih zahtjeva. Osim opširne službene dokumentacije korištenih alata, prilikom izrade aplikacije bit će korišteni YouTube tutorijali i web stranice, primjerice Stack Overflow. Teorijski dio rada ću potkrijepiti relevantnim dokumentima, knjigama te internetskim člancima.

Za pisanje dokumentacije korišten je Overleaf, alat za uređivanje teksta i kolaborativno pisanje znanstvenih dokumenata. IoT projekti biti će razvijeni uz pomoć ArduinoIDE alata kojim će se programirati razvojne ploče, dok će se alat Fritzing koristiti za izradu vizualne reprezentacije svakog IoT projekta. Firebase platforma u ovom radu služit će kao baza podataka i sredstvo komunikacije između IoT projekata i aplikacije. Mobilna aplikacija razvit će se korištenjem Dart programskog jezika u kombinaciji s Flutter okvirom. Naposlijetku, razvojna okruženja koja će se koristiti prilikom razvoja aplikacije su Microsoft Visual Studio Code te Android Studio.

3. Internet stvari

Internet stvari (eng. *Internet of Things*, skraćeno IoT) se može definirati kao skup fizičkih objekata, odnosno stvari, koji putem interneta izvršavaju svoju funkciju. "Stvari" mogu putem interneta biti međusobno povezane, mogu razmijenjivati podatke međusobno ili s nekim drugim sustavima. Zbog svoje fleksibilnosti, dostupnosti i jednostavnosti njihova upotreba eskponencijalno raste. U nastavku ovog poglavlja će se opisati glavne karakteristike i najvažnije primjene interneta stvari, njegove prednosti, ali i mogući problemi i nedostaci.

3.1. Povijest interneta stvari

Premda je pojam interneta stvari relativno nov, njegova popularnost i rastuća primjena učinile su da pojam vrlo brzo postane općepoznat. Stručnjaci tvrde da se internet stvari pojavio krajem prvog desetljeća 21. stoljeća, između 2008. i 2009. godine. Procjenjuje se da je u tom razdbolju broj povezanih uređaja na internet premašio broj ljudi povezanih na internet, čime se simbolično obilježava početak interneta stvari.

Unatoč tome vrijedi istaknuti najraniji oblik pametnog uređaja koji je razvijen čak 1982. godine na sveučilištu Carnegie Mellon u Sjedinjenim Državama. Riječ je o automatu Coca-Cole koji je putem ARPANET-a bio sposoban izvještavati o količini proizvoda u automatu kao i informirati o tome je li piće hladno ili nije [1]. No, prvo spominjanje pojma i njegova konceptualizacija pripisuje se Peteru Lewisu u jednom od njegovih javnih govora u rujnu 1985. godine. On je pojam interneta stvari definirao kao integraciju ljudi, procesa i tehnologije sa spojivim uređajima i senzorima čime se omogućuje daljinsko upravljanje i nadzor takvih uređaja [2].

Ideje i vizije međusobne integracije zasebnih uređaja i njihovo udaljeno upravljanje postoje odavno, a razvoj tehnologije i popularizacija interneta to su konačno i omogućile. Početkom 21. stoljeća pojam interneta stvari se počinje sve češće koristiti, a već kroz nekoliko godina se ubrzano razvijaju čitavi sustavi, sastavljeni od različitih tehnologija, koji ostvaruju sve potencijale interneta stvari.

3.2. Osnovne karakteristike

Kao što je već spomenuto, internet stvari se definira kao mreža povezanih fizičkih objekata koja zajedno sa softverom čini svrsishodan sustav. Stvari tako mogu međusobno komunicirati putem raznih tehnologija, a jedna od njih je i internet. O njemu će biti veći fokus u odnosu na direktnu komunikaciju između dva uređaja, takozvanu M2M komunikaciju (eng. *Machine to Machine*). Ovakvi sustavi mogu biti iznimno veliki, sačinjeni od velikog broja integriranih elemenata, koji se upotrebljavaju u različite svrhe, a najčešće će biti i navedene u nastavku. Osnovna prednost ovakvog sustava je što "stvar" prima vrlo precizne informacije iz okoline, samostalno ih obrađuje i šalje gdje god je potrebno raznim protokolima. Sve se odvija bez ljudske interakcije, bez ljudskog faktora koji može donositi pogreške, biti manje precizan i vremenski ograničen. Ta "stvar" može biti senzor tlaka u automobilskoj gumi koji će upozoriti vozača u slučaju niskog

tlaka u gumama, može biti detektor dima i smrtonosnih plinova, životinja koja ima GPS lokator u svojoj ogrlici ili osoba s dijabetesom koja na ruci ima uređaj za mjerenje razine glukoze u krvi. Dakle, svaki objekt kojem se može dodijeliti IP adresa, odnosno svaki objekt koji ima sposobnost spojiti se na mrežu i slati podatke. Objektima se može dodijeliti i jedinstveni indentifikator ili oznaka (eng. *tag*) koja se može pročitati putem RFID (eng. *Radio Frequency Identification*) tehnologije. Postavljanje oznaka na razne predmete omogućuje korisniku da lako identificira i kontrolira označene stvari. RFID funkcionira bežično, pomoću elektromagnetskih polja, prilikom čega RFID čitač odašilje radio valove koji napajaju pasivne RFID oznake. Nakon što je oznaka aktivirana tim impulsom, ona odašilje digitalne podatke koji dalje služe za identifikaciju [3].

Mnogo je čimbenika koji utječu na razvoj interneta stvari; veliku ulogu ima bežična tehnologija, mikroelektromehanički sustavi (MEMS, eng. *Microelectromechanical systems*) te razvoj samog interneta [4]. Mikroelektromehanički sustavi sastavljeni su od mikroskopskih uređaja, naročito od uređaja s pokretnim dijelovima. Njihova veličina može biti u nanometarskim rasponima, a često se nazivaju i mikrostrojevi ili mikrouređaji [5]. Što se tiče razvoja interneta najvažniji je internetski protokol IPv6 (eng. *Internet Protocol version 6*) jer ima jako veliku moć adresiranja i transferiranja velikog broja podataka. IPv6 je relativno nov protokol koji je 2017. godine postao i internetski standard, zamijenivši svog prethodnika IPv4 [6]. Zamjena se dogodila zbog rapidnog rasta broja uređaja spojenih na internet i koji trebaju jedinstvenu IP adresu. Kapaciteti protokola IPv4 su vrlo brzo postali nedostatni pa je uvođenjem protokola IPv6 taj problem riješen.

3.3. Arhitektura

Arhitekturu interneta stvari možemo razdvojiti u tri sloja: percepcijski sloj, mrežni sloj i aplikacijski sloj. Prvi sloj podrazumijeva povezane fizičke uređaje, senzore, RFID oznake, akutatore (dijelovi stroja koji ostvaruju fizičke pokrete). U ovom najnižem sloju odvija se prikupljanje informacija izvana od strane senzora i kontrola uređaja, dijelova stroja, akutatora i slično. U nekim slučajevima može se izostaviti proces slanja podataka na oblak ili server ako je potrebna što brža reakcija uređaja na očitanje senzora. U tom slučaju se komunikacija odvija direktno između senzora i uređaja koji će obaviti neku radnju, a podaci se sukladno tome obrađuju na licu mjesta (eng. *"edge" processing*).

U mrežnom sloju odvija se slanje prikupljenih podataka na internet uz pomoć računala, bežičnog ili žičnog pristupa na mrežu i drugih načina. Prikupljeni podaci se prije slanja pretvaraju iz analognog u digitalni oblik, agregiraju i formatiraju, a ovaj sustav prikupljanja i obrade podataka prije slanja se označava kraticom DAS (eng. *Data Aquisition Systems*). Ovaj sloj se u nekim slučajeima može i proširiti ako je potrebno pretprocesiranje podatka na licu mjesta (eng. *edge pre-processing*). Količine prikupljenih podataka u ovom stadiju mogu biti izrazito velike, pogotovo ako je riječ o industrijskoj proizvodnji sa stotinama senzora koji prikupljaju i simultano šalju gomile podataka. Postupkom pretprocesiranja bi se količina podataka filtrirala i komprimirala što olakšava kasnije slanje na server, oblak ili nešto slično. Ovdje do izražaja dolazi strojno učenje (eng. *machine learning*) i umjetna inteligencija. Zbog toga se u stvarnom vremenu može poslati povratna informacija stroju direktno te instantno poboljšati proces bez

da se čekaju instrukcije iz nekog udaljenog centra ili oblaka. Veliku ulogu u mrežnom sloju ima već spomenuti protokol IPv6 koji može podržati veliku skalabilnost interneta stvari. Također, jedan od najčešćih protokola koji se koriste je i MQTT protokol (eng. *Message Queuing Telemetry Transport*). Služi za komunikaciju između pametnih uređaja, pripada kategoriji laganih protokola, a najčešće koristi TCP/IP model za funkcioniranje.

Posljednji sloj je aplikacijski sloj gdje se odvija analiza dobivenih podataka, na temelju rezultata obrade i analize se donose odluke i kontroliraju uređaji, a podaci se skladište. Uz ovaj sloj se vežu inteligentni sustavi s podatkovnim centrima ili oblacima. Time se dobiveni podaci s različitih lokacija mogu integrirati u neku širu sliku, pomoći prilikom donošenja potrebnih akcija ili poslovnih odluka. Dakle, u ovom sloju se događa analitika podataka, menadžment i arhiviranje. Sustav može biti i u takozvanom oblaku, stoga se oblak može zamisliti kao mozak, sve uređaje i senzore kao udove, a mrežne protokole kao kralježnicu ili okosnicu tijela interneta stvari. Sustavi bazirani na oblaku dizajnirani su da skladište, procesiraju i analiziraju ogromne količine podataka te da koriste strojno učenje čime pružaju mnogo bolji uvid u stanje stvari. Nešto što se nikada ne bi moglo ostvariti spomenutim procesiranjem na licu mjesta. Takozvani "cloud computing" se sve više koristi u industrijskom internetu stvari (o kojem će biti riječi u jednom od sljedećih poglavlja) zbog povećanja produktivnosti, smanjenja zastoja i vremena čekanja, a pogotovo zbog uštede energije.

3.4. Uređaji i senzori

U ovom poglavlju bit će navedeni i opisani neki od najpopularnijih uređaja i mikrokontrolera, s naglaskom na one uređaje koji su se koristili prilikom izrade praktičnog dijela ovog rada. Uz njih, spomenut će se i najčešće korišteni senzori, servo motori i ostali elementi koji se koriste zajedno uz mikrokontrolere.

Pojam interneta stvari donedavno je zvučao prilično daleko, posao kojim se mogu baviti samo stručnjaci, čiji proizvodi se proizvode samo u velikim tvornicama. Međutim, internet stvari postao je izrazito dostupan svima zahvaljujući popularizaciji malih pločica (eng. *boards*) koje služe za razvoj vlastitih rješenja ili razvoj prototipova. Te pločice su veoma jeftine, lako se mogu programirati te nije potrebno veliko predznanje za njihovim rukovanjem. Tako se proteklih godina razvila velika zajednica koja se bavi razvojem zanimljivih rješenja pomoću tih malih računala, a sukladno tome i dio tržišta koji proizvodi te uređaje i ostalu elektroniku koja dolazi uz njih, neki od njih prikazani su i u nastavku.

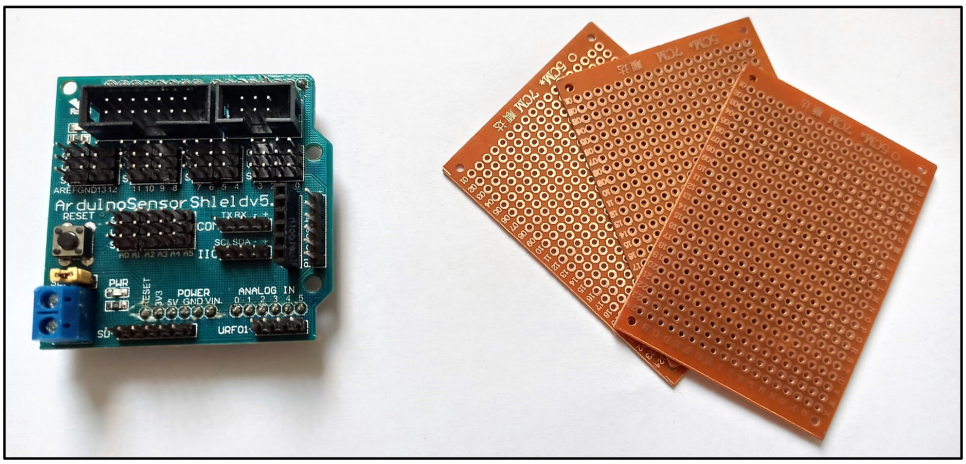
3.4.1. Arduino

Arduino je tvrtka koja proizvodi mikrokontrolere i dodatke za razvoj digitalnih uređaja i pametnih stvari. Arduino se može definirati kao elektronička platforma koda otvorenog tipa (eng. *open-source*) koja se temelji na hardveru i softveru jednostavnom za korištenje. Tvrtka Arduino je nastala početkom 21. stoljeća u Italiji, u gradu Ivrea. Ondje je na institutu *Interaction Design Institute Ivrea* student Hernando Barragán 2003. godine napravio razvojnu platformu *Wiring* za svoj diplomski rad, a Massimo Banzi i Casey Reas su ga mentorirali. Cilj diplomskog

rada bio je napraviti jednostavan i ekonomičan alat za razvoj projekata i digitalnih rješenja koji ne bi koristili samo inženjeri i stručnjaci, nego i studenti, početnici, hobisti i sl. Projekt se sastojao od PCB ploče (eng. *printed circuit board*) na kojem je bio mikrokontroler ATmega168. Za softverski dio pobrinuo se Casey Reas razvivši preteču današnjeg ArduinoIDE alata, platformu *Processing*, kojom se mogao programirati mikrokontroler. Na njihovim temeljima, uz doprinos ostalih kolega, se vremenom razvila tvrtka Arduino. Ime je dobila po talijanskom kralju Arduinu od Ivreje, ali indirektno, jer je isto ime nosio i kafić u kojem su se osnivači tvrtke redovno sastajali [7]. Imena razvojnih ploča također imaju veze s državom iz koje su potekla pa se tako među Arduinovim proizvodima nalaze imena poput Galileo, Leonardo, Diecimila, Uno, Due,...

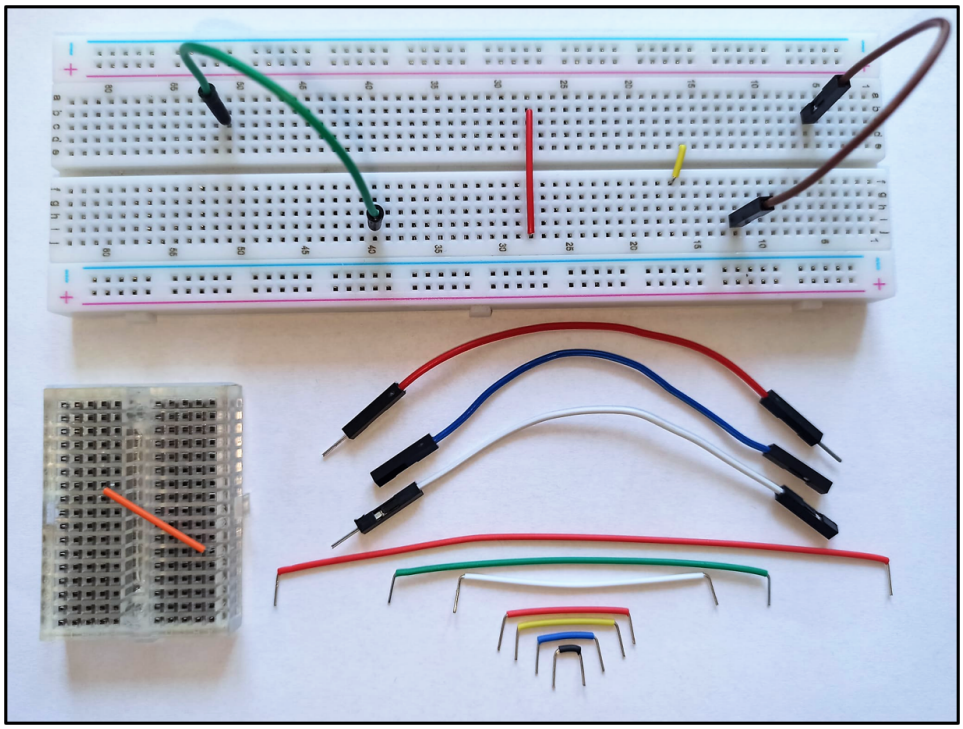
Kada se govori o Arduino hardveru bitno je pojasniti nomenklaturu i definirati što je što. Prvo treba ustanoviti značenje "mikro" pojmova: mikročip, mikroprocesor i mikrokontroler. Mikročip (skraćeno: čip) je mikroskopski set strujnih krugova, tranzistora, otpornika i ostalih elemenata smještenih na maloj pločici silicija. Mikročip je općenitiji pojam, dok su mikroprocesor i mikrokontroler sustavi koji se nalaze na mikročipu. Mikroprocesor je središnja procesorska jedinica, CPU (eng. *Central Processing Unit*), ugrađena na jedinstveni čip. To je mozak sustava i sposoban je izvoditi osnovne aritmetičke, logičke i ulazno/izlazne operacije. Međutim, to je sve što može raditi, procesirati podatke. Mikrokontroler nudi mnogo više jer osim CPU-a sadržava memorijske jedinice (ROM, RAM, EPROM,...), serijske portove, periferne ulazno/izlazne portove i ostale potrebne elemente što ga čini cjelovitim računalom na samo jednom čipu. Što se tiče procesne moći, mikroprocesor je mnogo snažiji jer je to njegova jedina svrha, stoga se koristi u računalnim sustavima. Dok se mikrokontroler zbog svoje jednostavnosti i samostalnosti koristi za specifične namjene i ugradbene sustave (eng. *embedded systems*).

Tehnički gledano, Arduino ploča nije mikročip ni mikroprocesor niti mikrokontroler, nego razvojna ploča bazirana na nekom mikrokontroleru. Precizan naziv bio bi mikrokontrolerska razvojna ploča (eng. *microcontroller development board*). Svaka ploča ima određen broj pinova (tzv. muški nastavci) ili utora u koje pinovi ulaze (tzv. ženski nastavci). Oni, između ostalog, služe za spajanje ploče na druge ploče za proširenje, takozvane štitove (eng. *shields*). *Shield* je posebna ploča dizajnirana tako da sa svoje donje strane ima "muške" pinove koji točno odgovaraju "ženskim" pinovima Arduino razvojne ploče na koju se onda jednostavno nadogradi, prikazana na slici 2. Oni pružaju razne funkcionalnosti osnovnoj ploči kao što je spajanje na internet, kontroliranje motora, povezivanje sa zaslonom, omogućavanje bežične komunikacije poput Bluetooth tehnologije,... Osim *shieldova*, pinovima se ploča povezuje sa posebnom verzijom matične ploče za razvoj prototipa (eng. *Breadboard*), ali i sa drugim sličnim elementima poput PCB ploče (eng. *Printed Circuit Board*). PCB služi za povezivanje elektroničkih komponenti koristeći provodne materijale na neprovodnoj površini, tj. ploči [8]. Većina PCB ploča dolazi s već ucrtanim provodnim stazama i elementima, međutim, postoje i ploče pogodne za razvoj prototipa sa slike 1 na koje se mogu lemiti razni elementi po želji i potrebama korisnika.



Slika 1: *Shield* i PCB ploče (Izvor: autorski rad)

Breadboard je ploča koja ima određen broj utora, ovisno o veličini, koja služi za povezivanje razvojnih ploča sa senzorima i ostalim elementima. Unutar ploče se nalaze metalne trake koje povezuju sve pinove jednog retka. Ne zahtijeva lemljenje stoga se može višekratno koristiti, a glavna uloga je prilikom razvijanja prototipa. Prigodna je za učenje pa se koristi u školama, fakultetima i raznim radionicama za početnike. Za spajanje *breadboarda* s Arduinom i ostalim elementima koriste se žice s posebnim nastavcima koje se lako ubadaju u utore mikrokontrolera i u utore *breadboarda*. Hrvatski naziv im je žičani kratkospojnici, ali inače se nazivaju *jumper wires* (ili *DuPont* žice). One ovisno o svom nastavku mogu biti muško-muške, muško-ženske, žensko-ženske.

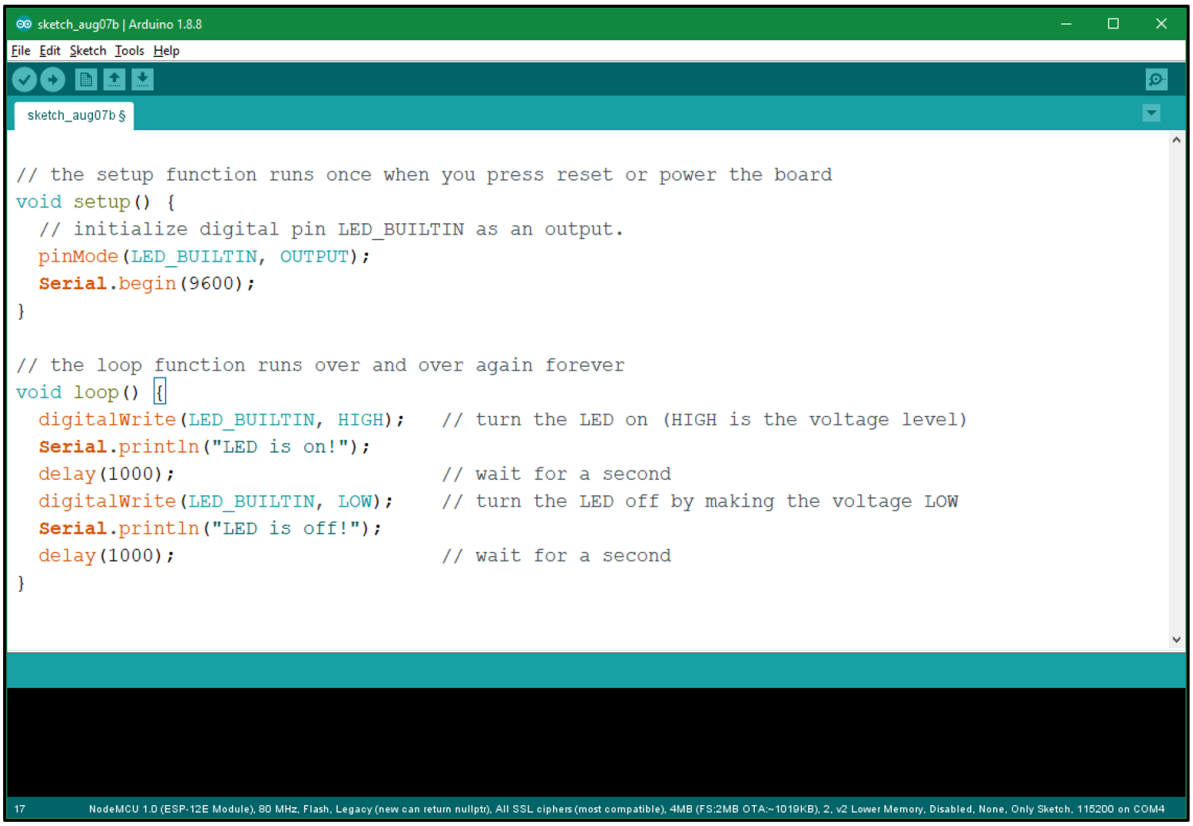


Slika 2: *Breadboardi* i žičani kratkospojnici (Izvor: autorski rad)

Pinovi se nazivaju ulazno-izlazni (skraćeno: I/O pinovi) zato što preko njih mikrokontroler prima signale i informacije iz okoline i/ili šalje naredbe. Naredba je također izlazni signal koji preko pina odlazi u druge dijelove sustava, primjerice aktivira motor robotske ruke, prikazuje podatak na OLED monitoru ili ga šalje na oblak putem interneta. Ponašanje mikrokontrolera predodređeno je uploadanjem programskog koda na uređaj od strane korisnika. Za to se koristi posebno razvijeno Arudino programsko okruženje, intuitivno za korištenje, dok je kod baziran na C++ jeziku (uz nekoliko posebnih pravila).

Arduino Software IDE (eng. *Integrated Development Environment*) je programsko okruženje otvorenog koda pisanog u Java jeziku, namijenjeno za višeplatformsku (eng. *cross-platform*) upotrebu (Windows, MacOS i Linux). Glavni element je sučelje za uređivanje teksta sa standardnim funkcijama poput kopiranja, ljepljenja, rezanja, pretraživanja i zamjene teksta, kao i automatskog indentiranja, sintaktičkog isticanja pomoću različitih boja i vizualne pomoći prilikom rada s vitičastim zagradama (od velike pomoći kada se u kodu nalazi mnogo uvjetnih blokova i petlji unutar petlji). Najvažnije funkcije aplikacije su opcije za kompiliranje koda i uploadanje koda na mikrokontroler. Prva opcija će izvršiti kod liniju po liniju i javiti preko ugrađene konzole eventualne sintaksne ili druge pogreške u kodu (logičke pogreške kompajler ne može detektirati). Druga opcija će napraviti isto što i prva uz dodatak "snimanja" upravo kompilirane skripte na mikrokontroler. Nakon uploadanja mikrokontroler izvršava skriptu dok god ima izvor električne energije ili dok se ne uploada nova skripta. Struktura koda za Arduino uređaje sastoji se od dva bloka, odnosno od dvije obavezne funkcije: *setup* i *loop*. One će unaprijed biti napisane čak i prilikom prvog pokretanja ArduinoIDE programa. Prva funkcija se izvršava samo jednom, prije svih ostalih funkcija, odmah nakon pokretanja uređaja ili uploadanja nove skripte. U ovom bloku se obično odvija inicijalizacija raznih elemenata, spajanje na servise, bazu podataka, na internet, sve ono za što je dovoljno da se odradi samo na početku. U drugoj funkciji zapisane su instrukcije koje će mikrokontroler zauvijek ponavljati, kao naprimjer očitavanje temperature svakih nekoliko sekundi i prikaz podataka na ekran. Na početku skripte, prije navedenih funkcija, se obično uključuju sve potrebne biblioteke i definiraju varijable koje će se koristiti. Skripta može sadržavati i mnoge druge funkcije osim *setup* i *loop*, kao i inače. Skripta napisana u Arduino IDE uređivaču sprema se lokalno na računalo kao tekstualna datoteka ekstenzije ".ino". Određeni broj biblioteka dolazi zajedno s aplikacijom kao standardne biblioteke, ali veliki broj napravljen je od trećih stranaka i developera čime se znatno povećava fleksibilnost i smanjuje ograničenost. "Tuđe" biblioteke su besplatne za korištenje, developeri ih dobrovoljno dijele s drugima u svrhu stvaranja bolje zajednice. U mnogo slučajeva nastanu tako da developeri naprave biblioteku kako bi ostvarili vlastiti projekt te ju u konačnici ostave i drugima na korištenje. Uz Arduino IDE dolazi i skup gotovih skripti odličnih za početnike, za vježbanje i testiranje. Neke od gotovih skripti ne zahtijevaju nikakvo spajanje žica na pločicu, kao što je paljenje i gašenje integrirane LED žaruljice na razvojnoj ploči ili standardni ispis teksta "Hello World!" na serijski monitor. Inače, serijski monitor se često koristi prilikom programiranja kao pomoć u testiranju i debugiranju. To je direktna veza između računala i Arduina zahvaljujući kojoj se mogu slati i primati poruke. Serijski monitor se otvara u posebnom prozoru, kao samostalan terminal, prilikom čega je bitno odrediti točan serijski ulaz (eng. *serial port*) kako bi se komunikacija mogla ostvariti. Serijski ulaz se određuje u skripti, unutar *setup* funkcije posebnom naredbom, npr. "Serial.begin(9600)". Kako je vidljivo na slici 3 otvoren je

port za komunikaciju na brzini prijenosa od 9600 (eng. *baud rate*).



Slika 3: Arduino Software IDE sučelje (Izvor: autorski rad)

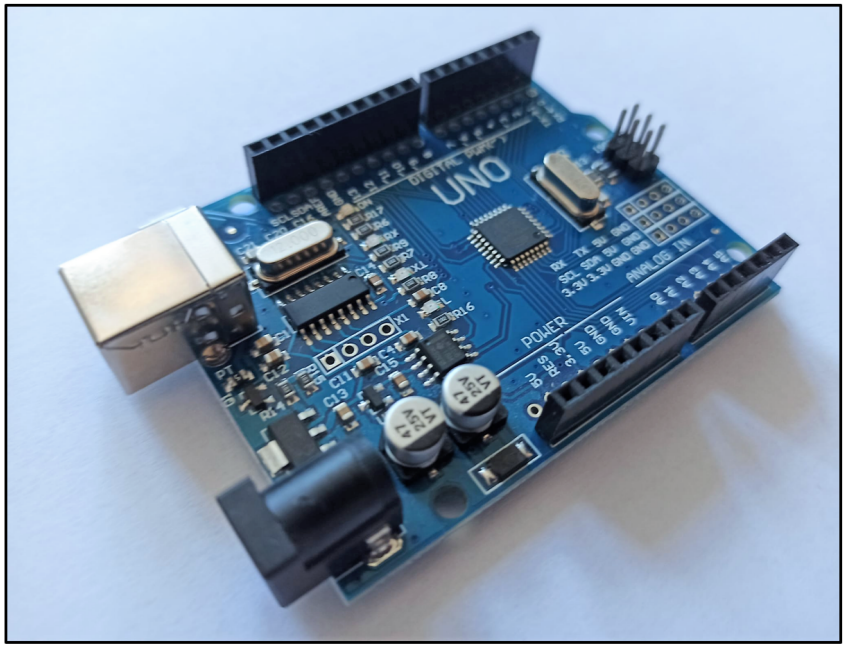
Otvaranje *porta* u početku omogućuje korištenje funkcije "Serial.println()" u nastavku programa. Funkcija prihvaća parametar u obliku teksta koji se zatim prikazuje na serijskom monitoru. Često korištena naredba je "digitalWrite()" koja šalje određeni signal prema navedenom pinu ili ugrađenoj LED žarulji. U gore prikazanom slučaju se prvo šalje signal za visoki napon koristeći drugi parametar "HIGH". Prvi parametar rezerviran je za mjesto na koje signal treba ići, u ovom slučaju signal se odašilje LED-ici koja se označava oznakom "LED_BUILTIN". Da se radilo o jednom od pinova (na kojeg je također mogla biti spojena LED-ica) tada bi se prvo morala definirati konstantna varijabla i pridružiti joj broj točnog pina koji je spojen na žarulju (npr. const int ledPin = 13). Ugrađena lampica samo je morala biti inicijalizirana u *setup* segmentu ovog koda s naredbom "pinMode()". "PinMode()" također treba dva parametra, prvi se odnosi na točan pin ili ugrađeni element, a drugi parametar označava je li taj element izlaznog tipa (poput lampice) ili ulaznog tipa (poput senzora temperature). Vjerojatno najčešća naredba koja se koristi u programiranju Arduina je "delay()". Naredbom "delay()" se određuje vremenski period u milisekundama u kojem se program neće izvršavati, odnosno nastavak izvođenja programa će biti odgođen za uneseni vremenski period. Poželjno je koristiti pauze naročito unutar *loop* funkcije zato što se ona, poput *while* petlje, jako brzo repetira. Stoga se prigodnim pauzama mikrokontroler može rasteretiti, naročito kod složenijih radnji koje i same mogu potrajati poput slanja podataka putem interneta.

Osim što se putem ArduinoIDE programiraju Arduino mikrokontroleri, moguće je pro-

gramirati mnoge druge kontrolere i razvojne ploče raznih proizvođača zahvaljujući doprinosima velike zajednice i trećih stranaka. Treće stranke razvijaju "jezgre" za nove ploče. Svaka vrsta ploče zahtijeva svoju "jezgru" (eng. *core*) kako bi mogla biti programabilna i kompatibilna s Arduinovim programskim okruženjem i bibliotekama. Zahvaljujući tim doprinosima moguće je povezati mikrokontroler drugog proizvođača s Arduinovim softverom nakon jednostavne instalacije odgovarajuće "jezgre" putem ugrađenog alata *Boards Manager*.

Tvrtka Arduino službeno komercijalno proizvodi 17 tipova razvojnih ploča, a za dizajniranje nekih od njih zaslužne su i američke tvrtke SparkFun Electronics i Adafruit Industries [7].

Arduino Uno je prva razvojna ploča tvrtke Arduino koja podržava komunikaciju putem USB tehnologije. Smatra se odličnim izborom za uvod u svijet Arduina i programiranja razvojnih ploča zbog jednostavnosti korištenja pa je sukladno tome i najpopularnija ploča iz Arduinovog asortimana. Glavni element i "mozak" ploče je mikrokontroler ATmega328P tvrtke Atmel. Radi na frekvenciji od 16 MHz, posjeduje 32KB programske memorije, 2KB radne memorije (RAM, eng. *Random-Access Memory*) te funkcionira na voltaži od 5V [9]. Na ploči se nalazi 14 ulazno-izlaznih digitalnih pinova i 6 analognih pinova, njihova svrha je povezivanje razvojne ploče s drugim elementima poput senzora, zaslona, motora, releja itd. Uno se može lako povezati s odgovarajućim *shieldovima* u svrhu proširivanja funkcionalnosti što je već prethodno objašnjeno.

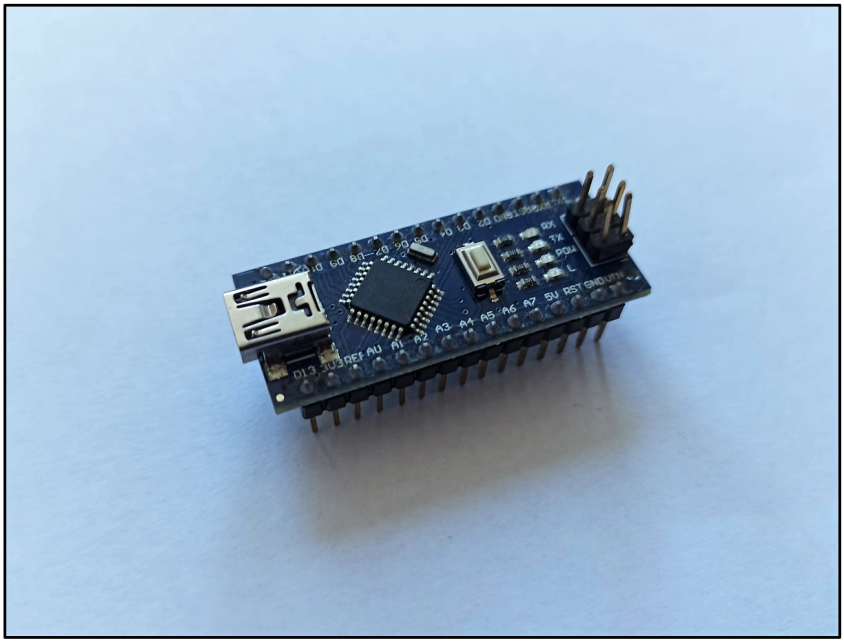


Slika 4: Arduino Uno (Izvor: autorski rad)

Osim analognih i digitalnih pinova, Arduino Uno sadrži dva pina za opskrbljivanje drugih elemenata potrebnom voltažom, a to su 5V pin (koji pruža reguliran napon od 5V) i 3V3 pin (koji pruža napon od 3.3V). GND pin služi za uzemljenje, a VIN se koristi za opskrbljivanje ploče naponom kada se koristi neki drugi izvor električne energije osim USB priključka i utora za napajanje. Ova ploča sadrži ugrađenu LED diodu, odličnu za testiranje i prvo korištenje,

i tipku *Reset* za ponovno pokretanje Arduina i ponovno izvršavanje programa uploadanog na ploču. Arduino Uno se programira putem USB kabela tipa B koji se obično dobije zajedno s razvojnom pločom, a isti utor služi i za napajanje. Drugi veliki utor na ploči je utor isključivo za napajanje (eng. *power jack* ili *barrel jack*), bez mogućnosti programiranja čipa. Služi za napajanje putem baterije ili AC/DC adaptera. Uno ima integriran sustav za regulaciju koji napon drži pod kontrolom kada je spojen s vanjskim uređajima.

Arduino Nano se može opisati kao smanjena, kompaktna verzija spomenutog Arduino Una zato što isto sadrži ATmega328 mikrokontroler, ima iste memorijske performanse i jednak broj analognih i digitalnih pinova. Nano nema utor za napajanje ni USB tipa B, ali zato koristi mnogo manji utor za USB Mini-B kablove preko kojeg se odvija programiranje i napajanje. Nije prikladan za povezivanje sa *shieldovima* zbog svoje veličine i zbog toga što nema setove "ženskih" pinova kao Uno. Umjesto toga, Nano ima setove "muških" pinova, koji su predviđeni za izravno postavljanje na spomenute *breadbordove* ili čak PCB ploče na koje se jednostavno mogu zalemiti (Slika 5). Obzirom da su veličinom otprilike pet puta manji od Una, prigodni su za korištenje u kompaktnijim projektima koji zahtijevaju što manju veličinu ili težinu.



Slika 5: Arduino Nano (Izvor: autorski rad)

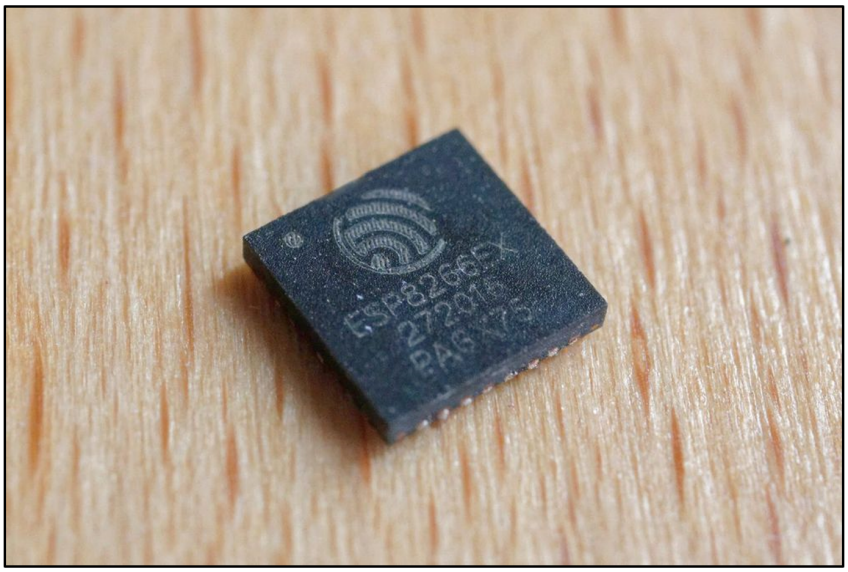
Od ostalih razvojnih ploča vrijedi spomenuti **Arduino Due** kao jednu od najvećih ploča, ali i prvu Arduinovu ploču kojom upravlja ARM procesor. Memorijski kapaciteti i frekvencija su višestruko veći od spomenutih Uno i Nano ploča dok je broj digitalnih pinova čak 54, a analognih 14. **Arduino Mega 2560** je veličinom i brojem pinova jednak kao Due, ali koristi ATmega2560 procesor slabijih performansi od prethodno spomenutoga. S računalom se povezuje putem USB tipa B priključka, odgovara standardnim *shieldovima* što ga čini odličnim odabirom za kompleksnije projekte koji zahtijevaju veliki broj analognih, a naročito digitalnih pinova (npr. projekti koji sadrže veliki broj gumbova). Ploča koja se izgledom razlikuje od gotovo svih ostalih je **Arduino Esplora** zbog svog eliptičnog oblika i već ugrađenih elemenata zbog

kojih podsjeća na *gamepad*. Esplora posebno odgovara korisnicima koji žele preskočiti proces učenja o elektronici i spajanja žica te odmah krenuti u programiranje i razvoj rješenja. Razlog tome je što u sebi već ima veliki broj ugrađenih elemenata, ulaznih i izlaznih. Sadržava LED diode koje emitiraju svjetlo i mali zvučnik što spada u kategoriju izlaznih elemenata (svjetlosni i zvučni). Od ulaznih elemenata ima *joystick*, nekoliko gumbova, *slider*, senzor za temperaturu, brzinomjer, mikrofon, senzor svjetlosti, a i *socket* za spajanje na LCD zaslon u boji. A kod ploča jedinstvenog izgleda ističe se **Arduino LilyPad**, savršeno okrugla razvojna ploča veličine kovanice. Posebno je dizajnirana kako bi se mogla ušiti u odjeću posebnom konduktivnom niti koja provodi struju, u svrhu ostvarivanja tzv. pametne odjeće ili e-tekstila.

3.4.2. Bežični moduli

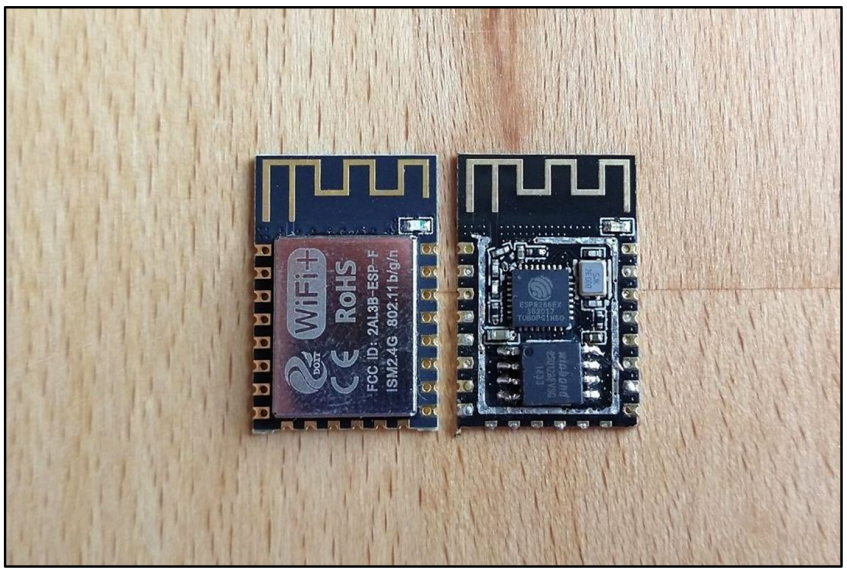
Postoje mnogi moduli razvijeni u svrhu bežične komunikacije putem interneta ili Bluetootha poput ESP-12E WiFi modula. Može ga se povezati s razvojnim pločama poput Arduina i tako omogućiti mikrokontroleru povezivanje s internetom. Unatoč tome što se razvojne ploče ručno mogu povezati s takvim modulima, moguće je dobiti i gotovu razvojnu ploču s već ugrađenim WiFi modulom spremnu za korištenje i programiranje.

ESP8266 je u suštini mikrokontroler s integriranim WiFi softverom. Budući da sadrži sve komponente računala na jednom čipu spada u kategoriju sustava na čipu (skraćeno SoC, eng. *System on a Chip*). Proizveden je od strane kineske tvrtke Espressif Systems sa sjedištem u Šangaju. Sastoji se od 32-bitnog, jednojezgrenog mikroprocesora *Tensilica L106* s taktom od 80MHz i WiFi primopredajnika [10]. Pregršt je verzija i inačica ovakvih čipova na tržištu zbog svoje raznovrsne namjene. Mogu biti integrirani u razne module koji se mogu nabaviti kao zasebni, samostalni moduli. A mogu biti i ugrađeni u razvojne ploče i slične sustave, ovisno o svrsi.



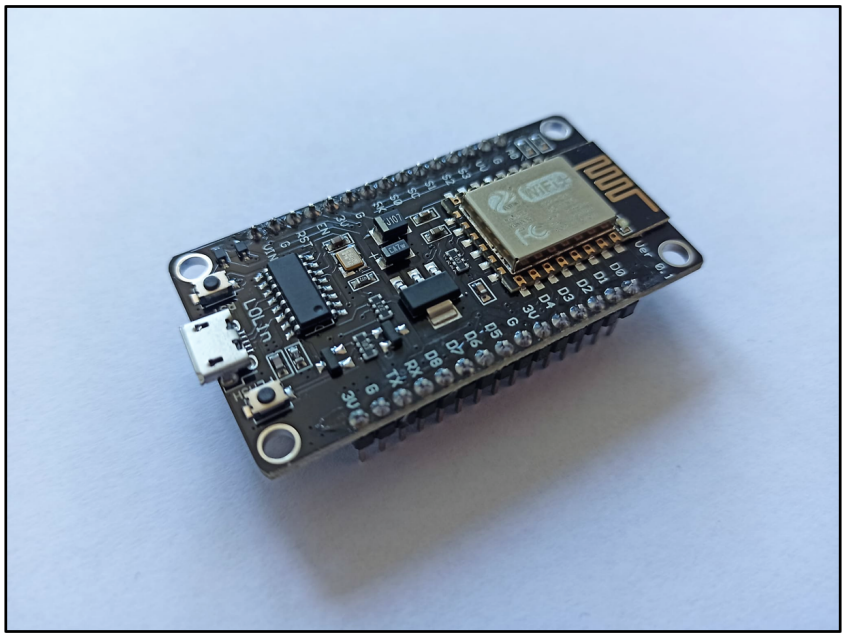
Slika 6: ESP8266 mikrokontroler (Izvor: preuzeto s [11])

Samostalni moduli (često se nazivaju i "sirovi") koriste se kao proširenje razvojnih ploča poput Arduina. Primjerice, WiFi moduli integrirani u WiFi *shield* obogaćuju razvojnu ploču novim funkcionalnostima. Nemaju direktno sučelje preko kojeg se mogu spojiti s računalom (npr. USB port). No i direktno uploadanje koda na module može biti izvedivo, iako to nije primarna namjena takvih proizvoda. Jedan od najpoznatijih takvih modula je ESP-01 koji svojom funkcijom uvelike može poboljšati razvojnu ploču unatoč veličini od samo 24.75mm x 14.5mm. U sebi sadrži spomenuti ESP8266 mikrokontroler kojem omogućava pristup internetskoj mreži. ESP-01 zbog svog rasporeda izlaznih pinova nije prigodan za razvoj prototipa spajanjem na *breadboard* ili za nadograđivanje razvojne ploče u obliku *shielda*. Zato je razvijen model ESP-05 specifične namjene, da se koristi kao *shield* za Arduino ili sličnu razvojnu ploču. Postoje i popularni modeli ESP-12, ESP-12E i ESP-201 kao samostalni moduli te poseban Testboard koji služi isključivo za razvoj prototipa i testiranje. Na slici 7 može se vidjeti kako izgleda ESP12 modul. Vrlo sličan modul, ESP12-E, nalazi se na razvojnoj ploči korištenoj za izradu projekata u nastavku. Na desnoj strani je ESP-12, ali bez metalnog pokrova pa se može vidjeti i kako je integriran ESP8266 mikrokontroler na sam modul.



Slika 7: ESP12 modul (sa i bez metalne zaštite) (Izvor: preuzeto s [11])

Razvojne ploče bazirane na ESP8266 su zapravo prethodno navedeni samostalni moduli integrirani u kompaktnu razvojnu ploču, a rezultat se može opisati kao Arduino s pristupom internetu. Dolaze s USB priključkom tako da je postupak programiranja sličan onome kao kod Arduina. Jedina distinkcija bila bi dodatan korak prije samog početka programiranja, a to je instaliranje odgovarajuće jezgre alatom *Board manager* unutar Arduino IDE. Primjeri takvih ploča su NodeMCU V1.0, Wemos D1 Mini, Wemos D1 R2 te ploča korištena za praktični dio rada, LoLin NodeMCU V3 Board (Slika 8).



Slika 8: LoLin NodeMCU V3 razvojna ploča (Izvor: autorski rad)

Budući da je praktični dio rada ostvaren korištenjem upravo ESP8266 SoC-a, potrebno ga je podrobnije analizirati. Točnije, korištena je razvojna ploča LoLin NodeMCU V3 s integriranim ESP-12E WiFi modulom na kojem se nalazi ESP8266. Kratica MCU u prijevodu označava mikrokontrolersku jedinicu (eng. *MicroController Unit*. Ploča se se sastoji od 30 pinova, po 15 s obje strane ploče. Ima utor za micro USB, ugrađenu LED žaruljicu i dva gumba, jedan za resetiranje ploče i jedan za tzv. *flash*. Najbitniji pinovi preko kojih se ostvaruje komunikacija s ostalim modulima i senzorima jesu analogni i digitalni pinovi. NodeMCU ima jedan analogni pin (A0) i šesnaest GPIO (eng. *General Purpose Input/Output*) pinova. Od njih šesnaest, jedanaest može biti korišteno u svrhu digitalnih pinova. Na samoj ploči je jasno naznačeno devet digitalnih pinova oznakama od D0 do D8, dok su preostala dva označena s "RX" i "TX". Unatoč takvom načinu označavanja na ploči, svaki pin s ploče ima svoju jedinstvenu korespondirajuću vrijednost unutar samo ArduinoIDE okruženja. Svi pinovi i njihove ArduinoIDE vrijednosti su vidljivi na Tablici 1. Uparene vrijednosti možda nisu intuitivne, stoga je korisno imati to na umu upravo zbog primjera u praktičnom dijelu rada.

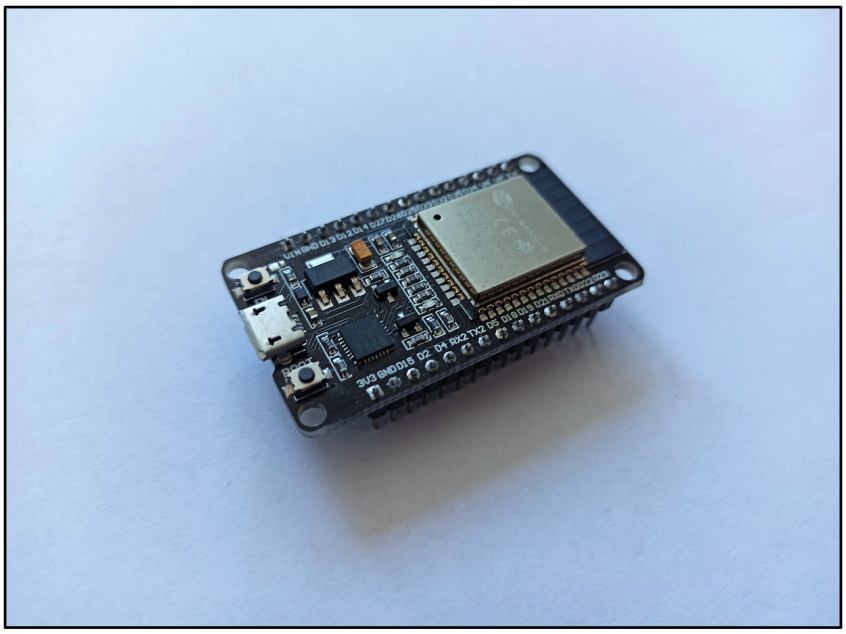
Tablica 1: GPIO pinovi i njihove vijednosti

Naziv pina na ploči	Vrijednost u ArduinoIDE	Alias u ArduinoIDE
TX	1	D10
D4	2	D4
RX	3	D9
D2	4	D2
D1	5	D1
D6	12	D6
D7	13	D7
D5	14	D5
D8	15	D8
D0	16	D0, LED BUILTIN
A0	A0	analog ip

(Izvor: Bonilla, 2019)

Primjeri povezivanja razvojne ploče s drugim elementima će kasnije biti prikazani pomoću dijagrama strujnog kruga (eng. *circuit diagram*) i autorskih fotografija. "GND" pinovi dolaze od riječi *ground* i služe za uzemljenje, ukupno ih ima četiri. "VIN" pin se koristi u slučaju da je potrebno razvojnu ploču i ostale povezane elemente napajati preko vanjskog izvora energije, a ne preko USB utora. "3.3V", za razliku od "VIN" pina, služi za napajanje ostalih elemenata povezanih s pločom. Ostali nisu toliko bitni te neće biti korišteni u praktičnom dijelu. Međutim, vrijedi spomenuti neke poput pina "EN", njime se ploča prebacuje u stanje minimalne potrošnje energije. Pin "RST" služi za resetiranje, a "WAKE" budi čip iz stanja dubokog sna.

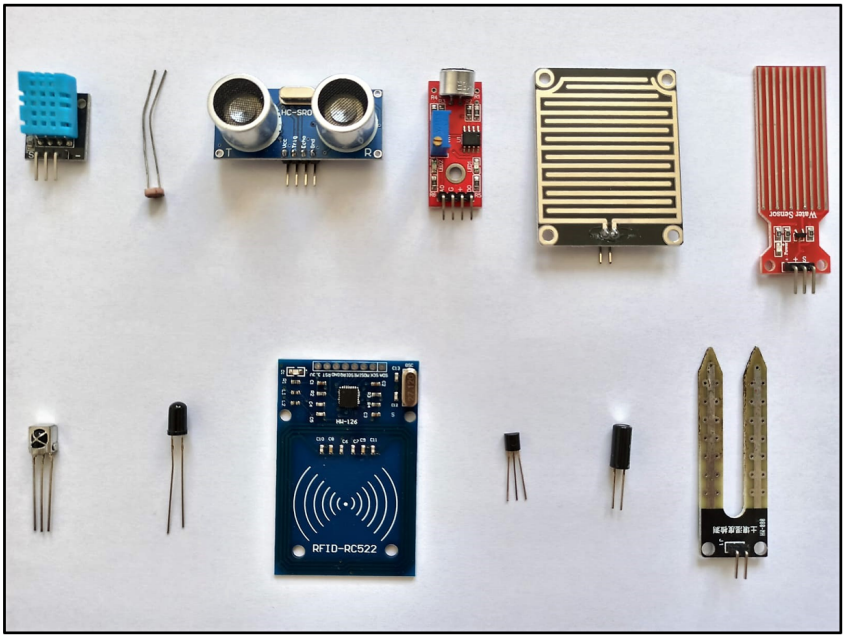
ESP32 je ekonomičan SoC s niskom potrošnjom energije i integriranim WiFi i Bluetooth funkcionalnostima. Nasljednik je i poboljšana verzija spomenutoga ESP8266. Nudi snažniji dvojezgreni procesor s dva do tri puta većom frekvencijom (160 do 240MHz). ESP32 nudi brži WiFi, više ulazno-izlaznih pinova, podržava Bluetooth verzije 4.2 i BLE (eng. *Bluetooth Low Energy*). Poput svog prethodnika, može se koristiti u obliku samostalnog modula koji nije praktičan za razvoj prototipa, testiranje, korištenje kao *shield* i povezivanje na *breadboard*. Mnogo praktičniji oblik je u kombinaciji s razvojnom pločom poput Adafruit ESP32 Feather, Sparkfun ESP32 Thing ili DOIT ESP32 DEVKIT V1 prikazan na slici 9. Sukladno navedenim poboljšanjima logično je za naslutiti kako je ESP32 nešto skuplji od prethodnika ESP8266, međutim obje razvojne ploče se mogu priuštiti za svega nekoliko dolara.



Slika 9: ESP32 (Izvor: autorski rad)

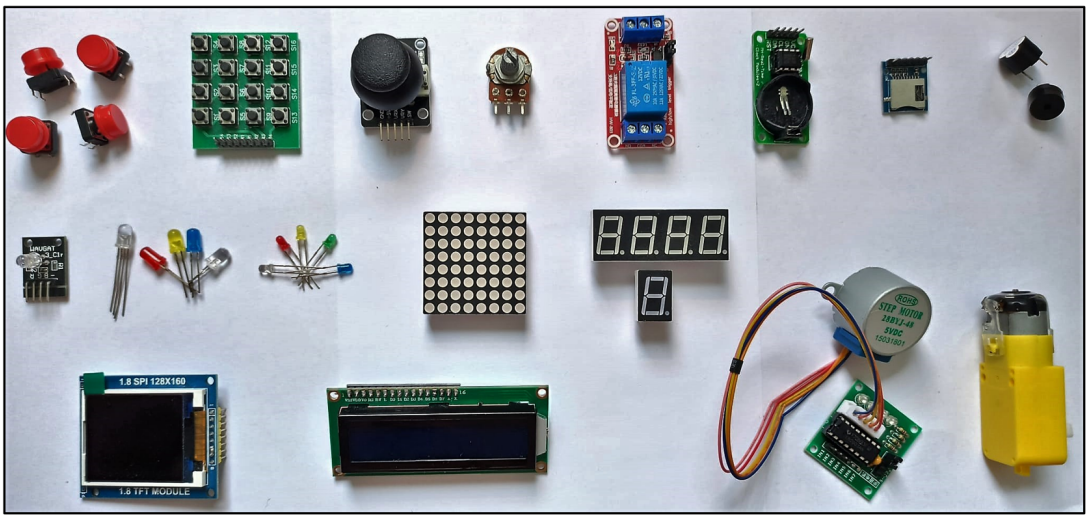
3.4.3. Senzori i ostali moduli

Paralelno s razvojem interneta stvari razvijaju se i mnogobrojni moduli i senzori čime se unaprijeđuje ova grana tehnologije i značajno povećavaju mogućnosti i primjene. Za međusobno povezivanje uređaja i modula potrebna je i odgovarajuća infrastruktura, ovisno o okruženju u kojem se koristi. WiFi je idealna tehnologija za mnoge primjene poput korisničkog korištenja (npr. pametni dom) ili komercijalnih primjena (npr. hoteli i restorani). Pruža domet od oko 100 metara što je sasvim dovoljno za navedene primjere, ali negativna strana je manjak sigurnosti i prilično velika potrošnja električne energije. Druga opcija je Bluetooth tehnologija koja je bitno ekonomičnija što se tiče potrošnje energije, ali pruža mnogo manji doseg. Prema tome, prikladna je za uređaje niske potrošnje koji ne koriste velike količine podatka, podržati će mnoge korisničke primjene, razna *fitness* i medicinska pomagala, rješenja za pametni dom, pametne zgrade i gradove. Spomenuta dva standarda su najpristupačnija i najčešće se koriste pogotovo kada je riječ o korisničkoj primjeni, dok industrije i komercijalna rješenja ponekad koriste i neke druge standarde. Jedan od njih je Zigbee, način za povezivanje uređaja niske potrošnje korišten u industrijama, rješenjima za automatizaciju i daljinsko upravljanje uređaja. Sigurniji od prethodnih tehnologija jer koristi AES (eng. *Advanced Encryption Standard*) enkripciju. Nema veći domet od WiFi-ja zbog čega nije prikladan u nekim industrijama koje tada sežu za rješenjima poput Lorawan tehnologije. Lorawan pripada skupu LPWAN (eng. *Low-Power Wide-Area Netowork*) komunikacijskih tehnologija, a zbog dometa od 15 kilometara i AES enkripcije prikladan je za komunikaciju uređaja u velikim industrijskim postrojenjima. Zaključno, mnogo je načina za ostvarivanje komunikacije u domeni interneta stvari što je potpuno razumljivo uzevši u obzir činjenicu da je lepeza primjene interneta stvari strahovito široka i neprestano se povećava.



Slika 10: IoT senzori (Izvor: autorski rad)

Putem senzora moguće je dobiti mnoge informacije poput temperature, položaja uređaja, brzine i slično čime senzori, u prenesenom značenju, postaju osjetila sustava. Senzor je prema definiciji objekt čija je uloga prepoznati događaje ili promjene u okolini na temelju kojih pruža odgovarajući odgovor [13]. Što se tiče primjene interneta stvari pomoću Arduina vrijedi navesti senzore koji se najčešće koriste u izradama digitalnih rješenja, a navedeni su redom prema slici 10. Senzor koji se pojavljuje u gotovo svim projektima za početnike je jednostavan DHT11 senzor za temperaturu i vlagu. Često se upotrebljava u kombinaciji s LCD zaslonom gdje se očitani podaci mogu prikazati, a postoji i skuplja, preciznija verzija DHT22. Fotootpornik je senzor za mjerenje intenziteta svjetlosti, a u svojoj biti je otpornik čiji otpor se smanjuje povećanjem intenziteta svjetlosti. Treći senzor po redu na slici 10 je ultrasonični senzor za udaljenost i može ga se često vidjeti na IoT projektima i robotima. Radi na način da odašilje zvučne signale, detektira ih ako se vrate natrag (ako se odbiju od prepreku), nakon čega se na temelju proteklog vremena izračunava udaljenost između senzora i objekta ispred. Senzor za zvuk detektira zvukove iz okoline, uz njega dolazi i integriran potenciometar kojim se može odrediti osjetljivost senzora. Pokraj njega nalazi se senzor za kišu i senzor za razinu vode koji su samoobjašnjivi, a o njima će biti riječi i kasnije budući da se koriste u praktičnom dijelu rada. U drugom retku senzora sa slike, prvi po redu je senzor infracrvenog svjetla koristan u projektima gdje se koriste daljinski upravljači, a obično i dolazi u paru sa simplificiranim daljinskim upravljačem. Nadalje, pokraj njega se nalazi senzor plamena koji izgledom podsjeća na crnu LED diodu. Plava pločica je RFID čitač (eng. *Radio-frequency identification*) široke primjene kako u korisničkim, tako i u komercijalnim i industrijskim svrhama. Preostao je još senzor za očitavanje isključivo temperature, valjkasti senzor za detektiranje vibracija i treskanja (eng. *shake switch*) te, naposlijetku, senzor za očitavanje vlažnosti tla. Ovi senzori se pretežito koriste za Arduino rješenja, ali većina se može koristiti i s drugim razvojnim pločama i mikroračunalima poput Rasberry Pi-a bez potrebe za većim modifikacijama.



Slika 11: IoT moduli (Izvor: autorski rad)

Spomenuti senzori se zbog svoje funkcije često nazivaju pasivnima jer se preko njih mogu samo dobiti informacije o događajima i promjenama u okolini. Zato postoje i mnogi drugi moduli s kojima se može interaktirati, koji mogu emitirati svjetlost ili zvukove, prikazati razne podatke i obavljati određene radnje. Na slici 11 prikazan je samo dijelić modula koji se mogu naći na tržištu pa će također redom biti imenovani i ukratko opisani. U gornjem lijevom kutu se nalaze četiri gumba, imaju po četiri "kukice" kojima se lako pričvrste na *breadboard* ili PCB ploču. Ako se radi o projektu koji zahtijeva veći broj gumba, odlično rješenje je 4x4 matrica pokraj. Veoma popularan modul je i komandna palica (eng. *joystick*) s dva ugrađena potenciometra, jedan za vertikalne i jedan za horizontalne kretnje, a upravo se jedan potenciometarski modul nalazi s desne strane. Potenciometar je u svojoj suštini otpornik čiji otpor može varirati okretanjem osovine, zahvaljujući tome se koristi za pojačavanje ili stišavanje (npr. mijenjanje jačine svjetlosti kod LCD zaslona). Crvena pločica s plavom kutijicom je tzv. relejski modul koji služi kao prekidač, otvara ili zatvara strujni krug, a detaljnije će biti obrađen u praktičnom dijelu rada. Sljedeći modul je modul za vrijeme i datum (eng. *Real time clock module*), zahtjeva malu bateriju te pruža informacije o trenutnom vremenu i datumu. Pokraj njega nalazi se microSD modul koji projektima omogućava dodatan izvor memorijskog prostora, ovisno o veličini microSD kartice. Posljednji moduli u prvom retku su mali, jednostavni zvučnici, korisni u alarmnim sustavima, tajmerima i slično. U drugom retku se nalaze LED diode u različitih boja i oblika, a prvi modul je RGB LED modul koji može emitirati svjetlosti raznih boja. U sredini je 8x8 matrica LED lampica i pokraj nje četveroznamenkasti i jednoznamenkasti svjetleći modul. Na dnu slike su dva zaslona drugačijih tehnologija, lijevo je OLED (eng. *Organic Light-Emitting Diode*), a desno je LCD (eng. *Liquid Crystal Display*) zaslon. U donjem desnom uglu su dva motora: koračni motor (eng. *stepper motor*) i tzv. TT motor. Oba pretvaraju električnu energiju u kinetičku energiju, odnosno rotaciju, pogodni za razvoj robotskih rješenja.

3.5. Primjena

Internet stvari se integrirao u gotovo sva područja života pa je i primjena veoma raznolika. Primjena se može podijeliti u nekoliko kategorija koje će u nastavku biti detaljnije opisane.

3.5.1. Korisnička primjena

Kategorija korisničke primjene obuhvaća najveći dio interneta stvari. Dovoljno je istaknuti podatak da je 2017. godine u svijetu bilo 27 milijardi IoT uređaja, a procjenjuje se da će taj broj 2030. godine iznositi 125 milijardi uređaja. Također se procjenjuje da 600 milijuna ljudi koristi pametne asistente glasovnim naredbama barem jedanput tjedno. Na temelju ovih brojki nije teško zaključiti o kakvom se tržištu radi. Konkretno, 2017. godine tržište interneta stvari iznosilo je 170 milijardi dolara te se očekuje iznos od oko 561 milijarde dolara samo pet godina kasnije, 2022. godine [14].

U kategoriju korisničke primjene pripada i najpoznatiji pojam vezan uz internet stvari, a to je pametni dom (eng. *smart home*). Pametni ili automatizirani dom je koncept koji obične uređaje u domu čini "pametnima" na način da ih poveže s internetom ili da direktno poveže više uređaja međusobno. To uključuje osvjetljenje prostora, grijanje i hlađenje, sigurnosne sustave, multimedijske sustave. Ovakav koncept omogućuje korisniku da upravlja elektroničkim uređajima s udaljenih mjesta, da uvijek ima pristup podacima koje senzori šalju na internet. Nije jednostavno ukratko opisati i generalizirati sve mogućnosti pametnog doma jer je primjena vrlo raznovrsna. Primjerice, korisnik može putem aplikacije ili neke platforme nadgledati potrošnju struje svakog uređaja u domu, ali i rolete se same mogu podizati u zoru ovisno o jačini svjetla koje očitava senzor. Osim olakšanog upravljanja domom i estetskih dodataka, važan segment je i sigurnost. Sustavi za sigurnost doma su zahvaljujući internetu stvari postali pristupačan i esencijalan dio pametnog doma. U prilog tome dovoljno govori činjenica kako se takvim sustavima služe milijuni ljudi diljem svijeta, a broj neprestano raste. Sigurnosnim sustavima korisnik može kamerama nadzirati dom bez prestanka, 24 sata dnevno. Pametnim bravama se može spriječiti neovlašten ulazak, alarmima spriječiti provalu te pametnim uređajima za praćenje spriječiti krađu. Osim toga, postoje i mnogi senzori koji korisniku mogu instantno javiti detektiranje dima, plamena, ugljikovog monoksida i drugih opasnih tvari. Osim samog doma, popularni su i uređaji koji se mogu koristiti i oko doma, u vrtu ili dvorištu. Riječ je o sustavima za automatsko navodnjavanje biljaka, praćenje ključnih čimbenika koji utječu na rast biljke kao što su temperatura, vlaga zraka, vlažnost tla itd.

Na tržištu već postoji mnogo gotovih sustava za implementaciju pametnog doma, pretežito od strane velikih kompanija sa svojim elegantnim rješenjima. Najbolji primjer tome je HomeKit tvrtke Apple koji omogućava korisnicima upravljanje uređajima putem iOS aplikacije ili glasovnim naredbama putem Siri asistenta. Također, tvrtke poput Amazona (Amazon Echo), Googlea (Google Home) i Samunga (SmartThings Hub) nude svoja rješenja. S druge strane, postoji i sve više nekomercijalnih sustava, softvera otvorenog koda (eng. *open source*) koji obuhvaća značajan dio tržišta.

Osim pametnog doma, jedna od glavnih korisničkih primjena je ona vezana za zdravlje.

Kombinacijom interneta stvari i zdravstvene industrije su stvoreni mnogi mehanizmi za poboljšanje kvalitete života. Tako je sada moguće neprestano motriti visinu tlaka, brzinu rada srca i slično te odmah dojaviti ako nešto nije u redu. Takvi sustavi su iznimno korisni kod ljudi sa zdravstvenim problemima, ali i kod rekreativnih ili profesionalnih sportaša. Na taj način moguće je neprestano nadzirati bolesne i starije osobe, a senzorima pada ili napadaja automatski javiti problem liječniku ili članu obitelji. Na tržištu postoje mnogi pametni satovi i narukvice sa spomenutim funkcijama očitavanja krvnog tlaka i brzinu rada srca. Pametni stetoskopi mogu direktno na mobitel mogu prikazati svoja očitanja. Postoje i pametni trudnički pojasevi koji mogu prepoznati ako nešto nije u redu s djetetom te mu čak i puštati glazbu.

Neki proizvodi interneta stvari korisničke primjene možda nisu korisni kao ranije navedeni primjeri, ali njihova popularnost i samo tržište mogu biti iznimno veliki. Takav primjer je industrija pametne odjeće koja u odjevne predmete integrira pametne uređaje. Pametnim čarapama se detektira dio stopala koji trpi najviše pritiska te podatke šalje u aplikaciju. Proizvodi se i odjeća za spavanje koja apsorbira toplinu i poboljšava kvalitetu sna. Postoji i pametna sportska odjeća, radna odjeća te gotovo svaki ostali odjevni predmet. Ako je potrebno navesti zbilja nekoristan primjer dovoljno je reći kako je tvrtka Pizza Hut proizvela je tenisice koje mogu naručiti pizzu.

Sustavi i rješenja interneta stvari korisničke primjene svakodnevno se povećavaju i razvijaju. Unatoč svim koristima koje pružaju, od iznimne je važnosti dobro se informirati o proizvodima ili sustavima, naročito zbog sigurnosnih rizika koje takvi prozvodi donose, a o čemu će biti govora u jednom od idućih poglavlja.

3.5.2. Komercijalna primjena

Komercijalna primjena interneta stvari podrazumijeva okolinu koju korisnik koristi svakodnevno izvan svog doma. Radi se o prostorima poput trgovačkih centara, poslovnih ureda, hotela, zdravstvenih ustanova, raznih prostora za zabavu i slično. Ovakvi sustavi pretežito služe za unaprijeđenje usluge i korisničkog iskustva, kao recimo u hotelima. Tržište za ovakvu vrstu primjene također se vremenom profiliralo, pogotovo kada se govori o malim, bežičnim uređajima niske potrošnje. Takvi uređaji su fleksibilni, skalabilni i veoma isplativi, omogućuju lako upravljanje te masovno prikupljanje i analizu podataka.

Uvođenje interneta stvari u komercijalne okoline omogućuje optimizaciju i automatizaciju cjelokupnog sustava. Također, čini lanac opskrbe pametnijim, korisnicima pruža kvalitetniju uslugu, ali i poboljšava sigurnost na radnom mjestu.

3.5.3. Industrijska primjena

Internet stvari u području industrije se odnosi na ekosustav u kojem se nalazi veliki broj međusobno povezanih i sinkroniziranih uređaja ili strojeva. Popularna je i kratica IIoT koja označava industrijski internet stvari (eng. *Industrial Internet of Things*), a ova nova eru industrije se naziva i "industrija 4.0" ili "industrijski internet". Industrijski internet definira se kao skup strojeva, računala i ljudi koji omogućuju inteligentne industrijske operacije koristeći

naprednu analitiku podataka za ostvarivanje povoljnih poslovnih ishoda [15].

Glavna karakteristika četvrte industrijske revolucije je korištenje takozvanih CPS-ova (eng. *Cyber-Physical Systems*) koji mogu međusobno komunicirati, donositi autonomne i decentralizirane odluke, utječući pri tome na povećanje efikasnosti, produktivnosti, sigurnosti i transparentnosti [16]. Prije same definicije CPS-a navest će se primjer iz stvarnog života kako bi se dobila prigodna predodžba o tome što to CPS uopće jest. Naime, u američkom sveučilištu MIT (eng. *Massachusetts Institute of Technology*) nalazi se vrt u kojem se uzgajaju rajčice o kojima se brine tim robota. Roboti informacije o biljci dobivaju putem senzora koje ima svaka biljka, na temelju kojih roboti sinkronizirano obavljaju potrebne radnje. Robot biljku može zaliti vodom i/ili gnojivom, oprašiti cvjetove, a kasnije može locirati i ubrati zrelu rajčicu [17].

Jednostavna definicija CPS opisuje kao sustav koji integrira fizičke sustave s kibernetičkim mogućnostima u svrhu poboljšanja njihovih fizičkih mogućnosti [18]. Dakle, kao što je navedeno u prethodnom primjeru, to je spoj fizičkih i računalnih mogućnosti koje zajedno mogu autonomno funkcionirati, primati i obrađivati podatke iz vanjskog svijeta, na temelju podataka stvarati virtualnu kopiju stvarnog svijeta, donositi odluke i obavljati fizičke radnje. Pojednostavljeno, uređaj ili stroj nakon četvrte industrijske revolucije postaje "pametan" i povezan na internet, stoga se naziva IoT uređaj. Ako taj uređaj služi u proizvodnji ili industriji, naziva se IIoT (eng. *Industrial Internet of Things*) uređaj. Više takvih uređaja tvori samostalne, inteligentne i automatske sustave koji zahtijevaju malo ili nimalo ljudske intervencije, a nazivaju se CPS (eng. *Cyber-Physical Systems*).

U današnje vrijeme gotovo sve industrije teže biti moderne, koristiti gore navedene koncepte i biti što efikasnije. Kao primjer jedne industrije može se navesti avionska industrija s tvrtkama kao što su Airbus ili Boeing. To su tvrtke koje proizvode ogromne zrakoplove sastavljene od više milijuna dijelova, a greške prilikom proizvodnje moraju biti minimalne ili nepostojeće. Tvrtka Airbus u svojim alatima i strojevima koristi razne senzore u cilju izbjegavanja potencijalnih nepravilnosti i poboljšanja sigurnosti, a sami radnici koriste nosivu tehnologiju, pametnu odjeću, kao što su pametne naočale. Takva tehnologija je samo u segmentu proizvodnje avionskih sjedala tvrtki Airbus ostvarila poboljšanje produktivnosti od petsto posto i broj pogrešaka reducirala gotovo do nule [19]. Tvrtka Boeing se isto tako prikazuje kao tvrtka budućnosti zbog modernizacije svoje proizvodnje. Primjerice, i Boeing je svoju proizvodnju obogatio proširenom stvarnosti zahvaljujući pametnim naočalama. Koriste ih električari prilikom spajanja i provlačenja žica jer su se dosada morale koristiti upute na papirima ili na laptopu. Pametne naočale su omogućile slobodne ruke (tzv. *hands-free*) i interaktivne dijagrame električnih vodova direktno pred očima, gotovo eliminirajući pogreške. Vojna industrija ima zanimljive primjene interneta stvari i već ukorijenjene kratice kao što su IoMT (eng. *Internet of Military Things*), IoBT (eng. *Internet of Battlefield Things*), OoT (eng. *Ocean of Things*). Slučajevi korištenja u vojnoj okolini tiču se ponajviše nadzornih sustava, izviđačkih akcija, nosivih biometričkih uređaja, ali i robota, vozila, senzora, streljiva te ostale tehnologije koja se koristi na bojištu. Ocean stvari (OoT) je projekt vođen od strane američke organizacije DARPA (eng. *Defense Advanced Research Projects Agency*) čiji je primarni fokus razviti mrežu interneta stvari preko cijelog oceana u svrhu kontrole i prikupljanja podataka o svim plovilima na oceanu, bila ona vojna ili komercijalna.

3.5.4. Infrastrukturna primjena

IoT se koristi u infrastrukturi ponajviše u svrhu sigurnosti i prevencije eventualnih problema. Koriste se u mostovima, željeznicama, vjetroelektranama i slično, gdje mogu detektirati ili predvidjeti nepravilnosti u strukturi i smanjiti sigurnosni rizik. Osim održavanja i kontrole, pruža mnoge prednosti prilikom samog procesa konstrukcije i izgradnje. Zahvaljujući velikoj količini podataka i analitici u stvarnom vremenu moguće je povećati efikasnost, produktivnost, uštedjeti resurse i uvelike pomoći pri donošenju odluka.

Primjena IoT-a u infrastrukturi temelj je i za ostvarivanje ideje pametnih gradova. Primjeri takvih gradova mogu se pronaći u Južnoj Koreji, Kini, Španjolskoj. U Španjolskom gradu Santanderu postavljeno je 20 000 senzora koji su povezani s gradskom aplikacijom, čije usluge koristi nekoliko desetaka tisuća građana. Aplikacija pruža informacije o slobodnim parkirnim mjestima, količini oborina u različitim dijelovima grada, gustoći prometa i slično. Osim toga uvedena je i pametna rasvjeta koja smanjuje emitiranje svjetlosti ako nema nikoga u blizini, pametno navodnjavanje parkova koje će se aktivirati samo onda kada je tlo suho, pametne kante za smeće koje šalju upozorenje smetlarima u trenutku kada se napune i slično [20]. Pametnim gradovima se tako smanjuje zagađenje okoliša, postiže se efikasnija proizvodnja i distribucija električne energije, racionalnija potrošnja vode i energenata, a u konačnici se nastoji poboljšati kvaliteta života.

3.6. Potencijal i budućnost

Za dobivanje prave slike o trenutnom i budućem stanju interneta stvari dovoljno je uzeti u obzir sljedeće podatke:

- 247 milijuna rezultata se dobije nakon upisivanja pojma "IoT" u Google tražilicu
- 127 uređaja se poveže na internet svake sekunde [21]
- 5 kvintilijuna bajtova/5 eksabajta/1 milijarda gigabajta se generira od strane IoT-a svakog dana [22]
- 80 posto industrijskih postrojenja koristi ili planira uvesti IoT uređaje [21]

Predviđa se:

- 596 milijardi eura prihoda vezanih uz IoT do 2022. godine [23]
- 41 milijarda IoT uređaja do 2027. godine [21]
- 4-11 trilijuna ukupnog udjela u tržištu do 2025. godine [21]
- 70 posto automobila povezano na internet do 2023. godine [21]

Gotovo je nevjerojatno do kojih razmjera se internet stvari razvio u nešto više od desetak godina, pogotovo s obzirom na činjenicu da je "rođenje" interneta stvari bilo između 2008. i 2009. godine. Ako je suditi po brojevima, koji su već sada impresivni, internet stvari nema granice. Infrastruktura postaje sve bolja, tehnologije se neprestano razvijaju, a mijenja se i način na koji čovjek funkcionira u ovom dinamičnom okruženju.

Tri su glavna čimbenika koja utječu na razvoj interneta stvari: umjetna tehnologija ili AI (eng. *Artificial Intelligence*), 5G mreže i velika količina podataka (eng. *Big Data*). Spomenute tehnologije su iznimno napredne same po sebi, a fuzijom tih elemenata stvara se temelj za razvoj moćnih, pametnih i automatskih sustava. AI postiže zastrašujuće rezultate, pomičući granice strojeva i uređaja koji danas imaju sposobnost učenja, adaptiranja, procesiranja informacija poput ljudskog bića. Povezivanjem umjetne inteligencije s internetom stvari stvara se pametan sustav međusobno povezanih uređaja koji putem 5G mreža komuniciraju gotovo u stvarnom vremenu. Ako se u tu jednadžbu doda i koncept *Big Data*, rješava se problem obrade velike količine podataka.

Pojmovi poput pametnih domova, gradova, industrije su već uvriježeni u svakodnevni govor, a budućnost već donosi mnoge inovitete. Sve se više ulaže u razvoj autonomnijih uređaja koji će moći obraditi sve veći broj podataka na licu mjesta, bez potrebe za slanjem na udaljene servere, oblake i slično. U tu kategoriju, između ostalog, pripadaju roboti i autonomni automobili. Sve su kvalitetniji i sustavi za obradu govora, tzv. NLP (eng. *Natural Language Processing*) te sigurnosni i nadzorni sustavi. Zbog brzine kojom se nove inovacije i tehnologije pojavljuju, teško je i predvidjeti kako će internet stvari i svijet izgledati u budućnosti.

3.7. Problemi i kontroverznost

Internet stvari, uz sve svoje prednosti koje omogućuje, sa sobom nosi i određene probleme:

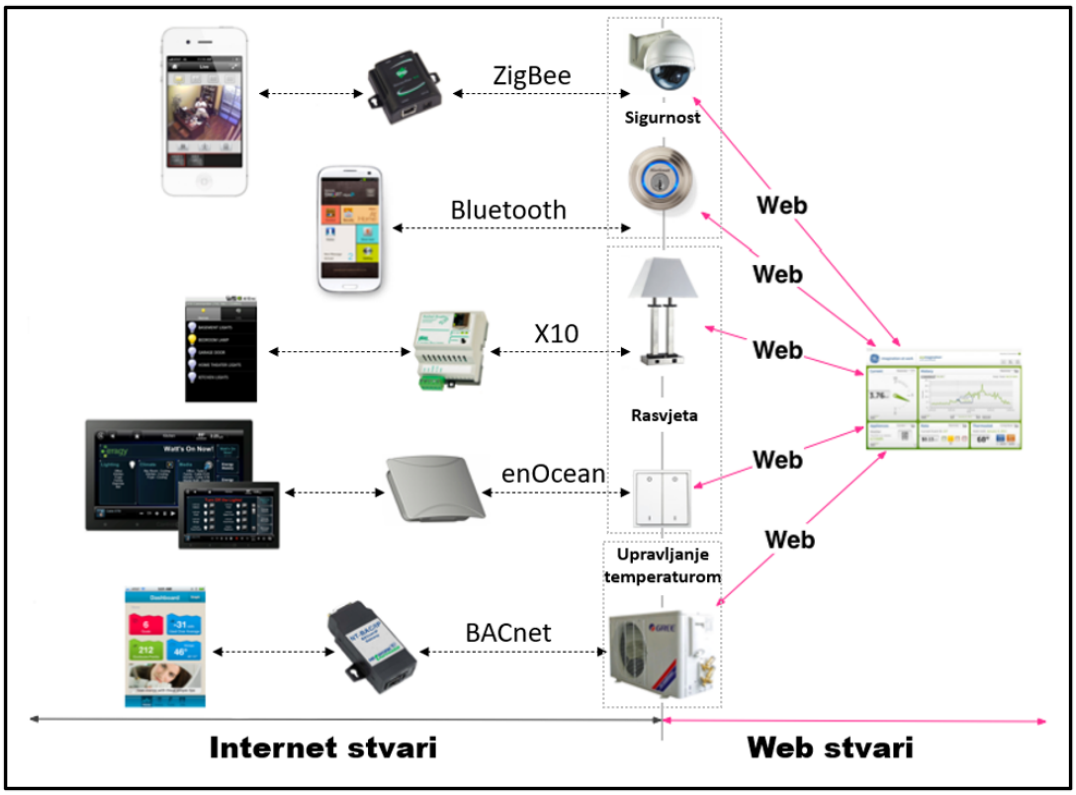
- Nedostatak stručnjaka i kvalificirane radne snage zbog prebrze ekspanzije interneta stvari
- Povišen rizik hakerskih napada zbog ranjivosti softvera i slabih sigurnosnih sustava
- Nepostojanje strogo definiranih standarda i problem interoperabilnosti zbog velikog broja različitih rješenja na tržištu
- Smanjenje privatnosti pojedinca zbog konstantnog nadzora i prikupljanja svih mogućih podataka

Većina navedenih problema dolazi direktno uz činjenicu da internet stvari nekontrolirano raste. Broj uređaja spojenih na internet se već broji u milijardama, konstantno se razvijaju nova hardverska, softverska i komunikacijska rješenja. Zbog toga je naivno misliti da se hakerski napadi neće događati i iskorištavati ovaj ranjiv sustav. U svojoj knjizi Mišak [24] objašnjava kako u vremenu pametnih telefona, automobila ili usisivača i njihova povezivanja kroz IoT, oni (pametni uređaji) postaju podložni hakerskim napadima, baš kao i mikročipovi u ljudima. Osim toga, tvrdi kako samo hakiranje mikročipa može omogućiti neovlašten pristup informacijama

pohranjenim na njemu, ugrožavajući tako privatnost, financijsko ili zdravstveno stanje korisnika. Upravo ugrožavanje privatnosti uzrokuje najviše skeptičnosti i paranoje kod sve više ljudi. A u modernim društvima privatnosti gotovo da i nema, naročito u pametnim gradovima. Mišak [24] za pametne gradove kaže kako su to najveći sustavi za prikupljanje podataka o svakom aspektu svakog pojedinca u povijesti, a krajnji cilj je ostvarivanje apsolutne kontrole. Navedene činjenice uzrokuju odbojnost kod pojedinca, a problem će se riješiti jedino uvođenjem regulativa u svrhu ograničenja invazije na privatnost, barem u vlastitom domu.

4. Web stvari

Pojmovi internet stvari i web stvari mogu zvučati poput sinonima, ali postoji bitna razlika među njima. Internet stvari je obrađen u prethodnim poglavljima iz kojih se može zaključiti da IoT nastoji povezati veliki broj različitih uređaja na različite načine. Zbog velikog broja proizvođača i proizvoda ponekad je potrebno koristiti i isto toliko različitih platformi i protokola. Primjerice, moguće je imati aplikaciju povezanu s nadzornom kamerom putem Zigbee protokola, posebnu aplikaciju za upravljanje pametnim žaruljama putem WiFi-ja, nekakvu drugu platformu za upravljanje potrošnjom struje itd. Ako korisnik, primjerice, želi pametni dom, najbolja opcija bi bila kupiti sve uređaje od istog proizvođača. Ta preopterećenost tržišta i manjak interoperabilnosti zahtijevaju rješenje. Veoma je kompleksno, gotovo nemoguće, napraviti jedinstveni komunikacijski protokol za efikasnu komunikaciju svih pametnih stvari. Vremenom su se razvijali takvi protokoli, čak i novi jezici, međutim vrlo brzo je postalo jasno da takav način zahtjeva previše vremena i resursa za standardizaciju. Pod tim okolnostima se stvorio koncept weba stvari, odnosno korištenje već postojećeg, afirmiranog sustava kao što je web. Zato se web stvari definira kao mrežni standard koji omogućuje komunikaciju između pametnih stvari i web aplikacija.



Slika 12: Razlika između IoT i WoT (Izvor: preuzeto i prilagođeno s [25])

Web stvari je premostio problem interoperabinosti i potrebu da svaki uređaj ima drugačiji protokol i vlastitu aplikaciju, kao što imaju uređaji na lijevoj strani slike 12. Oslanja se isključivo na protokole i alate aplikacijske razine, a gotovo svaki protokol ili standard interneta stvari može biti povezan na web zahvaljujući "mostovima" (eng. *gateways*). Obzirom da je WoT

fokusiran aplikacijski sloj, koji pruža visoku razinu apstraktnosti, moguće je povezati podatke i servise s mnogo uređaja koji zapravo imaju drugačije transportne protokole. Prema tome, WoT pristupom je lakše programirati uređaje, podaci i servisi se mogu brže povezati, jednostavnije je razviti prototipe, rješenja i, u konačnici, održavati velike sustave [25]. S druge strane, IoT pristup ima fokus na niže razine, optimiziraniji je za fiksne, ugrađene uređaje i kontroliranje potrošnje baterije, memorijskih i mrežnih performansi i sl.

Uređaji niske razine se pomoću WoT koncepta apstrahiraju na višu razinu. Kao što je web globalna platforma za distribuciju aplikacija putem interneta, tako je isti princip iskorišten za pretvaranje interneta stvari u web stvari. U webu stvari su kompleksnosti i razlike među protokolima konačno zanemarene, a pažnja je usredotočena na aplikacijsku logiku. Mnogo benefita se ostvaruje spomenutim pristupom. Moguće je koristiti moderne web standarde na uređajima, integrirati ih na web zajedno sa servisima, na isti način na koji bi se inače razvijala web stranica. Sukladno tome, sve aplikacije će komunicirati s uređajima kao kad bi komunicirale s web servisom koji koristi web API (eng. *Application Programming Interface*). Time se pruža mogućnost za razvoj novih vrsta interaktivnih aplikacija. Koristeći standarde poput HTML, CSS i JavaScript-a se na web stranicu lako mogu prikazati prethodno prikupljeni i obrađeni podaci.

Web standardi (HTTP, HTML, CSS, URL, JavaScript...) su prisutni već dugo vremena, otvoreni su i besplatni te garantiraju stabilnost. Omogućuju brzo i pouzdano distribuiranje podataka diljem mreže. Web sigurno neće najednom prestati raditi ili zahtjevati ažuriranje, dok s gomilom IoT protokola dolazi taj rizik. Značajna prednost WoT-a je velika neovisnost među pametnim stvarima koje se mogu individualno mijenjati i razvijati. Stari uređaji mogu komunicirati s novim uređajima bez potrebe za ažuriranjem, isto kako se danas još uvijek mogu posjetiti stare web stranice bez ikakvih problema.

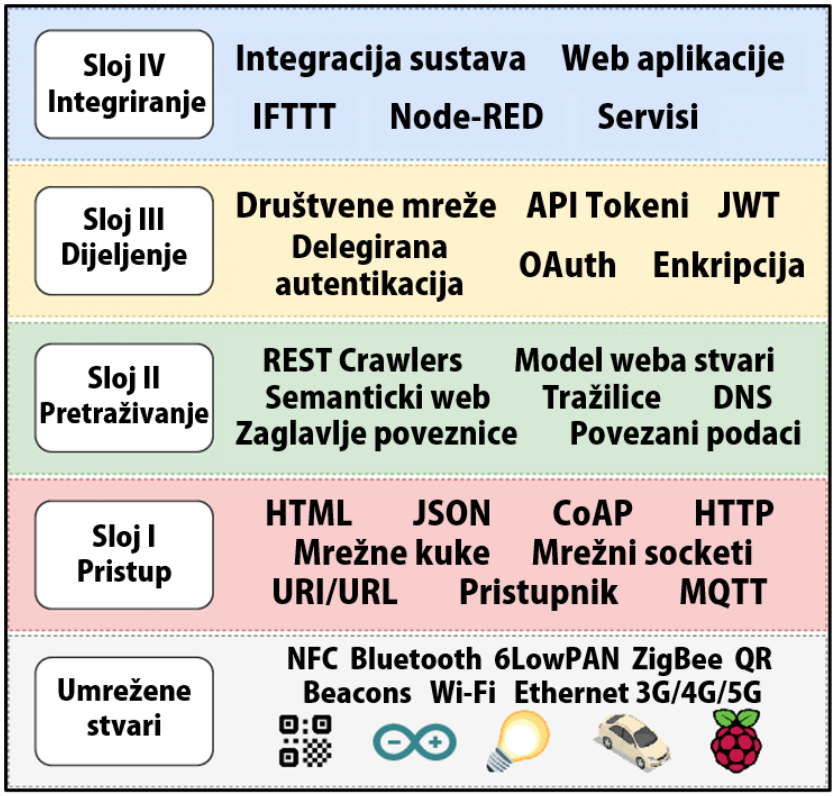
4.1. Arhitektura

Glavni elementi arhitekture weba stvari su korisnici. stvari i posrednici. Stvar je naziv za fizički ili virtualni entitet standardiziranih metapodataka koji ga opisuju [26]. Stvari moraju biti povezane na mrežni sustav kako bi mogle interaktirati putem mrežnog sučelja (eng. *interface*). Primjerice, na uređaju se može nalaziti HTTP server dok u pozadini senzori i ostali elementi mogu normalno komunicirati s fizičkim uređajem. Posrednici se uvode ako lokalnu mrežu nije moguće dohvatiti preko interneta. Zahvaljujući posredničkom WoT sučelju moguće je ostvariti komunikaciju između korisnika i stvari. Principi weba stvari se mogu upotrijebiti u svim razinama i načinima povezivanja elemenata kao što je veza između više stvari, veza između stvari i pristupnika (eng. *gatewaya*), veza između stvari i oblaka te veza između pristupnika i oblaka.

Svaka web stvar ima četiri arhitekturalna apsekta: ponašanje, interakcijske mogućnosti, sigurnosne postavke i povezivanje protokola [26]. Interakcijske mogućnosti pružaju model prema kojem se korisnik može povezati sa stvarima, a povezivanje protokola daje dodatne informacije za ispravno ostvarivanje komunikacije. Sigurnosnim aspektima se kontrolira pristup interakcijskim mogućnostima i samim podacima.

Arhitektura weba stvari je slična internetu stvari, a može se podijeliti u pet razina. Na

najnižoj razini nalaze se fizički ili virtualni uređaji, uključujući spomenuti Arduino, Raspberry Pi, pripadajuće senzorske sustave i ostalo. Druga razina je pristupna razina preko koje se ostvaruje komunikacija s uređajima. To su protokoli poput HTTP (eng. *Hypertext Transfer Protocol*), CoAP (eng. *Constrained Application Protocol*), MQTT (eng. *Message Queuing Telemetry Transport*) i drugi. Korištenjem ostalih web tehnlogija poput JSON (eng. *JavaScript Object Notation*), URL (eng. *Uniform Resource Locator*) te URI (eng. *Uniform Resource Identifier*) se ostvaruje bolja integracija cijelog sustava. Treća razina WoT arhitekture služi za pretraživanje i pronalazak umreženih objekata, a sastoji se od raznih algoritama i tehnologija poput semantičkog weba, pretraživača (eng. *search engines*), DNS-a (eng. *Domain Name System*) itd. Predzadnji sloj je zadužen za dijeljenje resursa, a primjeri takvih tehnologija su RESTful API mehanizmi i sigurnosni alati poput enkpripcije, OAuth standard za omogućavanje pristupa i API tokeni. U najvišem sloju su sustavi za pojednostavljen razvoj aplikacija, a najpoznatiji su IFTTT (eng. *If This Then That*), Node-RED i drugi servisi web aplikacija [27].



Slika 13: Arhitektura weba stvari (Izvor: preuzeto i prilagođeno s [27])

IFTTT servis je intuitivan alat za integraciju aplikacije s uređajima i drugim servisima. U suštini, funkcionira na principu okidača koji se aktiviraju na temelju radnje korisnika ili vanjskih događaja i promjena (npr. na temelju očitavanja senzora). Pruža korisniku mogućnost kreiranja mnogih ovisnosti između uređaja, servisa i aplikacije. Primjerice, korisnik ima senzor na zvonu ulaznih vrata i putem IFTTT platforme se lako može postići da korisniku stigne e-mail, obavijest na aplikaciji ili da se upali žarulja pred vratima svaki put kada netko pozvoni na vrata. Dakle, programira se određena akcija na temelju nekog događaja, odakle je platforma i dobila ime. Ovakav sustav je pogodan za razvoj pametnog doma, čak i za neiskusne korisnike. Node-RED

je *flow-based* alat vizualnog programiranja tvrtke IBM za povezivanje fizičkih komponenti, API-ja i servisa [28]. Alat je primarno dizajniran za razvoj IoT rješenja, ali lako se može prilagoditi raznim upotrebama zbog mogućnosti pisanja vlastitih JavaScript elemenata.

4.1.1. Funkcionalni zahtjevi

Web stvari zahtjeva neke principe i svojstva koji bi se trebali poštivati u standardnoj WoT arhitekturi. Arhitektura bi trebala podržavati međusobnu interakciju i "suživot" različitih sustava putem web tehnologija, a sama arhitektura bi trebala biti temeljena na RESTful API principima. Arhitektura mora podnijeti opterećenja karakteristična za web okruženje i varijacije među arhitekturama uređaja.

Četiri su glavna principa WoT arhitekture: fleksibilnost, kompatibilnost, skalabilnost i interoperabilnost. Flekibilnost se odnosi na mogućnost podržavanja svih varijanti arhitektura uređaja upravo zbog velike raznovrsnosti samih uređaja koji se svakodnevno pojavljuju na tržištu. Kompatibilnost označava težnju za podržavanjem starih i novih IoT standarda koji će međusobno funkcionirati i biti kompatibilni. Skalabilnost je važan princip za rješenja koja zahtijevaju tisuće ili milijune uređaja. Od uređaja se očekuje ostvarenje njihovih mogućnosti u WoT arhitekturi premda ne moraju biti napravljeni od strane istog proizvođača. Interoperabilnost je potrebna za ispravno funkcioniranje između uređaja i različitih *cloud* servisa. Tendencija je imati WoT uređaj bez da se mora znati proizvođač i arhitektura uređaja te mogućnost da se taj uređaj poveže s *cloud* servisom nepoznatog proizvođača bez potrebe za dodatnim radnjama.

Što se tiče samih stvari, odnosno uređaja, WoT arhitektura bi trebala omogućiti čitanje i ažuriranje njihovih podataka, pretplaćivanje na obavijesti o promjeni stanja uređaja, poticanje ulaznim ili izlaznim parametrima određenih akcija ili procesa. Također, arhitekrura bi trebala omogućiti pristup podacima, atributima i funkcionalnostima svake stvari te pretraživanje stvari prema navedenim elementima. Potrebno je podržavati i tzv. opisne mehanizme pomoću kojih se uređaji i njihove funkcije mogu opisati, ali na način da opis bude čitljiv i od strane računala. Opisivati se trebaju i atributi uređaja poput naziva, verzije, opisa samog uređaja i funkcija, poveznica s ostalim uređajima i ostalo.

WoT arhitektura treba podržavati web protokole koji se najčešće koriste, a to su protokoli korišteni na internetu i protokoli korišteni na lokalnoj mreži (eng. *Local Area Network*). Iste funkcionalnosti moraju biti pristupačne bez obzira na vrstu protokola koji se koristi, čak i ako se koriste kombinacije više protokola za funkcionalnosti istog uređaja. Uređaji na lokalnoj mreži bi također trebali biti dostupni i globalno. Model i hijerarhija sustava s više uređaja, korištenje *gateway* i *proxy* elemenata te virtualnih entiteta, to su različiti načini korištenja koje WoT arhitektura treba osigurati.

4.1.2. Tehnički zahtjevi

Apstraktna arhitektura weba stvari zahtijeva i ispunjenje određenih tehničkih uvjeta. Da bi pristup uređaju bio ostvariv potrebno je koristiti opis njegovih funkcija i sučelja, a naziva se opis stvari ili TD (eng. *Thing Description*). Opis stvari sadržava osnovne informacije o uređaju,

metapodatke, sigurnosne informacije, detalje o transportnom protokolu. Pod metapodatke se podrazumijeva jedinstveni identifikator uređaja (URI), serijski broj, datum proizvodnje i slične informacije. Sigurnosne informacije su vezane uz autentikaciju, autorizaciju i sigurnu komunikaciju. Opisi stvari su obavezni i moraju biti sadržani ili u uređaju ili nekoj vanjskoj lokaciji, ali moraju biti uvijek dostupni. Aplikacije biti trebale biti sposobne dohvatiti opise preko mreže te programirati sučelja koji ovise o tim opisima, odnosno metapodacima. Opisi fizičkih i virtualnih uređaja moraju biti dostupni vanjskim entitetima, imati opći način dijeljenja opisa, tj. TD-a.

Pristupačnost je stavka koja ne spada direktno u tehničke zahtjeve prema W3C (World Wide Web Consortium). Razlog tome je što ljudi, koji su uključeni u razvoj, su developeri koji integriraju uređaje s aplikacijom. Krajnji korisnici će imati interakciju samo sa sučeljem aplikacije ili sa sučeljem samog uređaja. To su dakle krajnji korisnici i oni ne pripadaju ovim WoT specifikacijama.

4.2. Korištenje

Za primjenu weba stvari u stvarnom životu se podrazumijeva prethodno napisano poglavlje o primjenama interneta stvari. Jedina je razlika što web stvari omogućava udaljeno kontroliranje svim uređajima, pristup uređajima putem web preglednika ili aplikacije.

Stoga će u ovom poglavlju biti opisani načini na koji se uređaji mogu povezati međusobno, s kontrolerima, serverima i slično.

- Najjednostavniji slučaj korištenja je lokalni uređaj kojim se upravlja daljinskim upravljačem (npr. klima uređaj). U ovom slučaju, daljinski upravljač bi se mogao implementirati kao gumb u aplikaciji ili web stranici, a komunikacija se odvija na razini lokalne mreže.
- Komunikacija između dvaju uređaja se najčešće postiže server-klijent vezom gdje jedan uređaj može slati poruku drugom i time izazvati neku akciju. Primjerice, klima uređaj je povezan sa senzorom temperature. Nakon što temperatura padne ispod određene granice, senzor to detektira, šalje poruku klima uređaju koji se zatim upali i radi dok se temperatura ne spusti.
- Slučaj korištenja sličan prvom je slučaj udaljenog pristupa. Tu korisnik može upravljati uređajem daljinskim upravljačem dok je unutar lokalne mreže, ali i izvan nje, samo nekim drugim protokolom. Kontroler se može nalaziti na pametnom mobitelu te može prelaziti s mobilne mreže na kućnu i obrnuto.
- Između interneta i kućne mreže se često postavlja posrednik, takozvani *gateway*, primjerice u pametnim domovima. *Gateway* upravlja uređajima unutar kućne mreže i može primati zapovijedi od strane daljinskog upravljača (npr. pametnog mobitela kao u prethodnom slučaju). Pomoću *gatewaya* se može dobiti i virtualna reprezentacija uređaja kao i dobrobiti koje inače pruža vatrozid (eng. *firewall*). U ovom slučaju, *gateway* je samo posrednik između lokalne mreže i interneta.

- U nekim slučajevima *gateway* ima sposobnost obavljanja određenih funkcija pa se naziva i *edge gateway*. Najčešće se upotrebljava u industrijskim rješenjima gdje može obaviti pretprocesiranje, filtriranje i agregiranje velikih količina podataka.
- Zahvaljujući *cloudu* ili serveru moguće je napraviti digitalne blizance stvarnih fizičkih uređaja te komunicirati s njima. To pruža prednosti kod uređaja koji nisu konstantno online, omogućuje provođenje simulacija i testiranje, a tek onda komunikaciju sa stvarnim uređajem.

4.3. Sigurnost

Sigurnost i privatnost su važni elementi koje se mora uzeti u obzir u svim implementacijama weba stvari. Neki općeniti problemi već su prethodno navedeni vezani uz internet stvari. Cilj je osigurati sustav čijim podacima mogu pristupiti samo autorizirani korisnici. Bitno je reći kako web stvari ne može pretvoriti nesiguran sustav u siguran, apsolutna sigurnost i privatnost ne može se garantirati, ali se arhitekturno mogu podržati sigurnosni mehanizmi. I u ovom pogledu je web stvari napredniji od interneta stvari zbog korištenja stabilnih i pouzdanih web protokola, ali još uvijek nijedan sustav nije neprobojan.

Uređaji mogu sadržavati osjetljive podatke zapisane u svojim TD-ovima u obliku metapodataka. Zato se nastoji postići strogo razdvajanje javnih i privatnih metapodataka. TD bi trebao posjedovati samo javne metapodatke kojima korisnici mogu prostupiti samo ako su autorizirani. Uređaji mogu posjedovati i informacije o identitetu osobe ili korisnika koji se mogu zlouporabiti. Takvi podaci trebaju biti minimizirani, a ako je moguće i sasvim izostavljeni. Generalno gledajući, TD-om bi se uvijek trebalo rukovati kao da posjeduju osjetljive identifikacijske podatke. Trebali bi se spremati i prenositi na što sigurniji način, pristup bi trebali imati samo autorizirani korisnici, trebali bi se brisati nakon određenog perioda. Također, WoT načini povezivanja moraju ispravno podržavati sigurnosne mehanizme koji se već nalaze na IoT platformama ili uređajima. Ove mjere mogu znatno smanjiti sigurnosne rizike.

U WoT okruženju potrebno je izolirati skripte koje se na uređaju izvode, a sadržavaju osjetljive osobne podatke. Ako se takve skripte odvoje smanjuje se mogućnost curenja podataka. Fizički uređaj također može biti ugrožen ako skripta postane neispravna ili kompromitirana, pogotovo ako sučelju uređaja nedostaju sigurnosne provjere. U takvom slučaju potrebno je minimizirati broj sučelja koja su izložena opasnosti. Kao rješenje tog problema koriste se dodatni hardverski mehanizmi koji mogu odbiti izvršavanje određenih naredbi koje bi mogle naštetiti uređaju.

Neki uređaji podržavaju ažuriranje sebe, svojih skripti i ostalih podataka uključujući sigurnosne vjerodajnice. To može biti slaba točka i prilika za napade. Da bi se rizik od napada smanjio potrebno je poštivati sigurnosne preporuke prilikom ažuriranja uređaja. Osim toga, veliki rizik je i spremanje vjerodajnica na uređaj. Kada bi napadač došao do njih mogao bi pristupiti povjerljivim podacima i izvesti hakerske napade poput DoS napada (eng. *Denial of Service*). WoT mora osigurati sigurno spremanje vjerodajnica, njihovu integriranost i povjerljivost.

5. Praktični dio

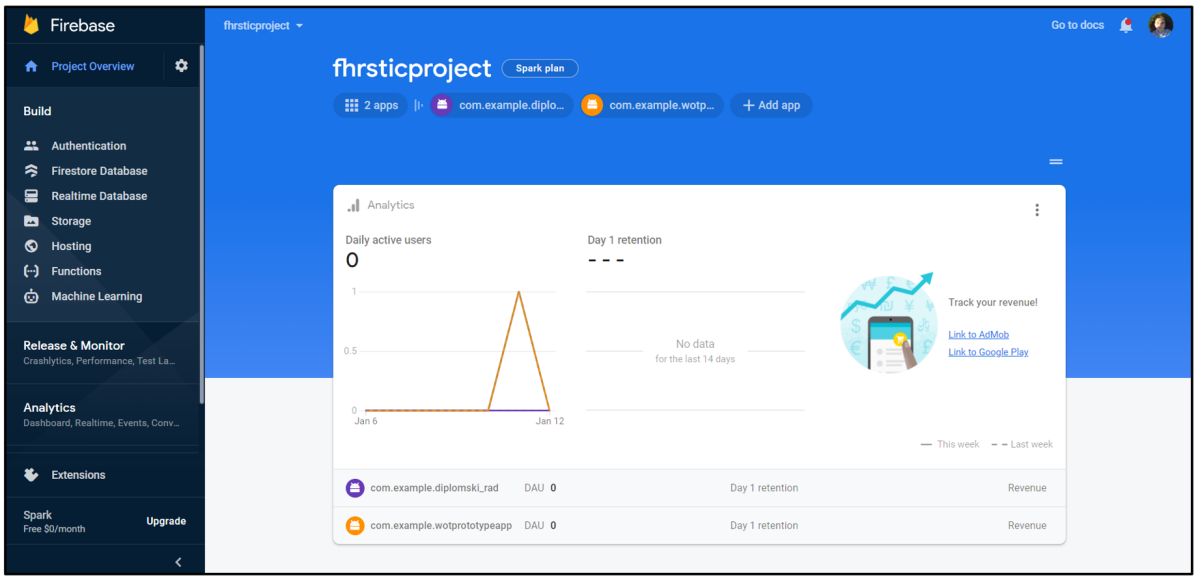
5.1. Korišteni alati i tehnologije

U nastavku su navedeni svi alati i tehnologije korišteni za izradu ovog rada, uz koje su priloženi i najvažniji dijelovi koda. Poglavlje je podijeljeno u pet cjelina: Firebase, Fritzing, Android Studio, ArduinoIDE i Flutter. ArduinoIDE se koristi za programiranje mikrokontrolera, Flutter za izradu mobilne aplikacije, a Firebase je "most" između njih. Fritzing je alat za izradu vizualno pristupačnih dijagrama kako bi se jednostavnije prikazao svaki zaseban projekt u ovom radu. Android Studio se koristi prilikom razvoja mobilne aplikacije jer omogućava testiranje na virtualnim Arduino uređajima.

5.1.1. Firebase

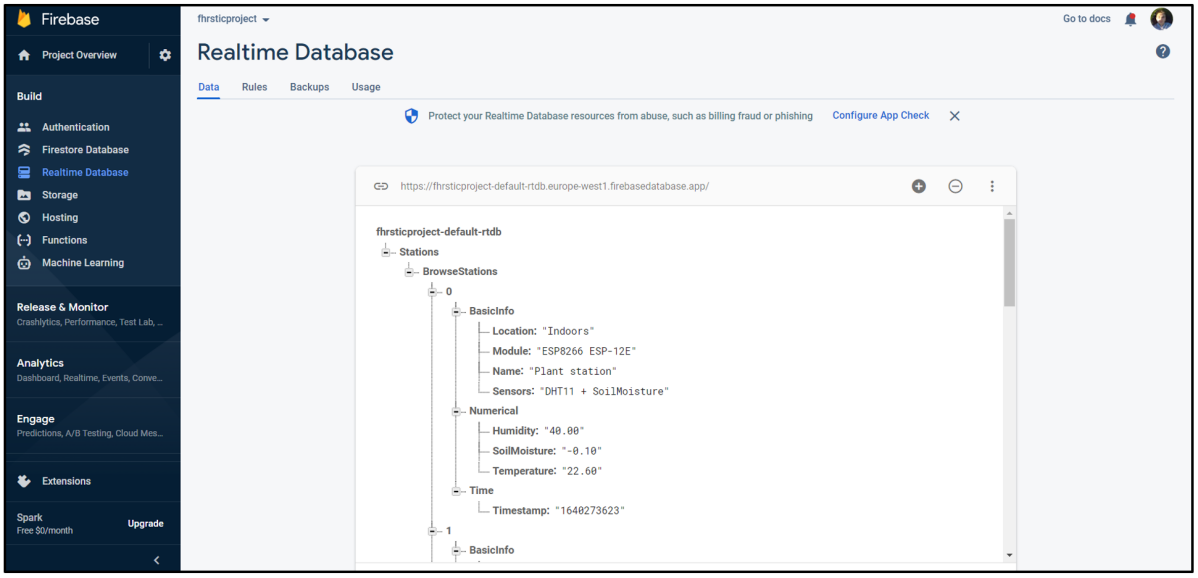
Firebase je platforma za razvoj mobilnih i web aplikacija, osnovana 2012. godine, a od 2014. godine je u vlasništvu tvrtke Google [29]. Firebase je takozvana BaaS (eng. *Backed-as-a-Service* platforma koja služi kao pružatelj *backend* servisa: autentikacijski servisi, pohrana podataka u oblak (eng. *Cloud Storage*), baza podataka u stvarnom vremenu (*eng. Realtime Database*), strojno učenje (eng. *Machine Learning*), *hosting* usluge, izvještavanje o padu sustava (eng. *Crash report*), itd. Produkt korišten za realizaciju mobilne aplikacije je *Firebase Realtime Database*. U prijevodu baza podataka u stvarnom vremenu, ima ulogu API-ja koji sinkronizira podatke podatke u Flutter aplikaciji s podacima na Firebase-ovom oblaku. Izvrsno rješenje softverskim programerima za razvoj aplikacija koje rade u stvarnom vremenu.

Postupak pripreme Firebase-a za korištenje započinje kreiranjem besplatnog korisničkog računa na službenoj Firebase stranici firebase.google.com. Nakon kreiranja računa, pokraj korisničkog avatara će se pojaviti gumb "Go to console". Klikom na gumb se pristupa konzoli sa svim korisnikovim projektima. Prilikom prvog pokretanja potrebno je napraviti novi projekt opcijom "Add project". Prvi korak je imenovanje novog projekta, a zatim i odabir Google-ove analitike ako je potrebna u projektu. Kreiranje novog projekta sastoji se samo od ta dva koraka, nakon kojih se otvara novi pogled "Project overview" (Slika 14).



Slika 14: Firebase - *Project Overview* (Izvor: autorski rad)

S lijeve strane se nalaze padajući izbornici sa svim mogućim funkcionalnostima koje pruže Firebase podijeljeno u četiri grupe: *Buid, Release & Monitor, Analytics* i *Engage*. Kako bi se funkcionalnosti mogle koristiti obavezno je pridružiti aplikaciju Firebase projektu odabirom opcije "Add app". Firebase podržava aplikacije za iOS, Android, Web i Unity platforme. Odabire se opcija Android čime započinje proces integracije Firebase-a s mobilnom aplikacijom od nekoliko koraka. Otvara se novi pogled s nekoliko obaveznih polja, a prvo je unos naziva Android paketa. Budući da se koristi Microsoft Visual Studio Code, taj naziv je automatski izgeneriran u kodu aplikacije. Može ga se pronaći u Visual Studiu, na putanji "android/app" u datoteci "build.gradle", kao vrijednost atributa "aplicationId". Ta vrijednost se može izmijeniti po želji ako korisnik ne želi koristiti generički naziv, bitno je samo da vrijednost bude jedinstvena. Potrebno je kopirati taj string i zalijepiti ga u polje "Android package name". Ostale opcije prvog koraka su opcionalne, uključuju unos nadimka aplikacije te unos certifikata potrebnog za korištenje "Google Sign-in" opcije i za opciju podržavanja telefonskog broja prilikom autentikacije. Prvi korak se završava gumbom "Register app" čime se otvara drugi korak. U drugom koraku potrebno je preuzeti datoteku "google-services.json" i postaviti je u isti direktorij projekta u kojem se nalazi i spomenuta "build.gradle" datoteka. Datoteka ima ulogu identifikatora tako da će mobilna aplikacija znati na koje *backend* Firebase servise se mora povezati. U idućem koraku se dobivaju isječci koda koje je potrebno kopirati i zalijepiti u određene datoteke u projektu. Time je Firebase spreman za korištenje, preostaje odraditi još nekoliko detalja sa strane Flutter aplikacije.



Slika 15: Firebase - *Realtime Database* (Izvor: autorski rad)

Produkt koji se koristi *Realitme Database* koji se odabere s lijeve strane sučelja, iz *Build* padajućeg izbornika. Baza podataka je zapravo datoteka tipa JSON, sastoji se od čvorova, a svaki čvor može imati neodređen broj djece. Koristeći se tim principom konstruirana je baza podataka kojom se služe svi Arduino projekti i Flutter mobilna aplikacija. Glavi čvor zove se "Stations" s jednim djetetom "BrowseStations". Djeca tog čvora su zapravo indeksi Arduino projekata, počevši od nula pa nadalje. Pod brojem nula je i prvi projekt čije informacije se kategoriziraju u tri cjeline: "BasicInfo", "Numerical" i "Time". Uloga djeteta "Time" i vremenska oznaka pod atributom "Timestamp" objašnjena je u nastavku, u ArduinoIDE poglavlju. "BasicInfo" za djecu ima sve osnovne podatke o projektu poput naziva projekta, korištenih senzora, podataka o razvojnoj ploči itd. Čvor "Numerical" sadrži brojčane podatke koje očitavaju senzori, razvrstane po vrsti. Postoji nekoliko vrsta podataka u Firebase bazi podataka, a grupirani su zbog lakšeg rukovanja kasnije u samoj aplikaciji. Podaci mogu biti brojčani ("Numerical"), tekstualni ("NonNumerical"), *boolean* ("Interactive") i posebni brojač ("Counter") za RFID projekt.

Firebase pruža mnoge druge funkcionalnosti, no za funkcioniranje ove mobilne aplikacije dovoljne su mogućnosti baze podataka u stvarnom vremenu. Konfiguriranje kvalitetnog "stabla" uvelike olakšava komunikaciju između baze i aplikacije, ali i Arduino skripti. Dakle, podaci u bazi će se gotovo automatski mijenjati sukladno radnjama korisnika u aplikaciji, a redovito će se i ažurirati zahvaljujući novim podacima koje šalju svi uređaji. Time je ostvaren temeljni zahtjev ovog rada, a to je da svi uređaji, odnoso razvojne ploče, budu povezane na Web i da se svima njima može upravljati preko istog protokola, preko iste aplikacije.

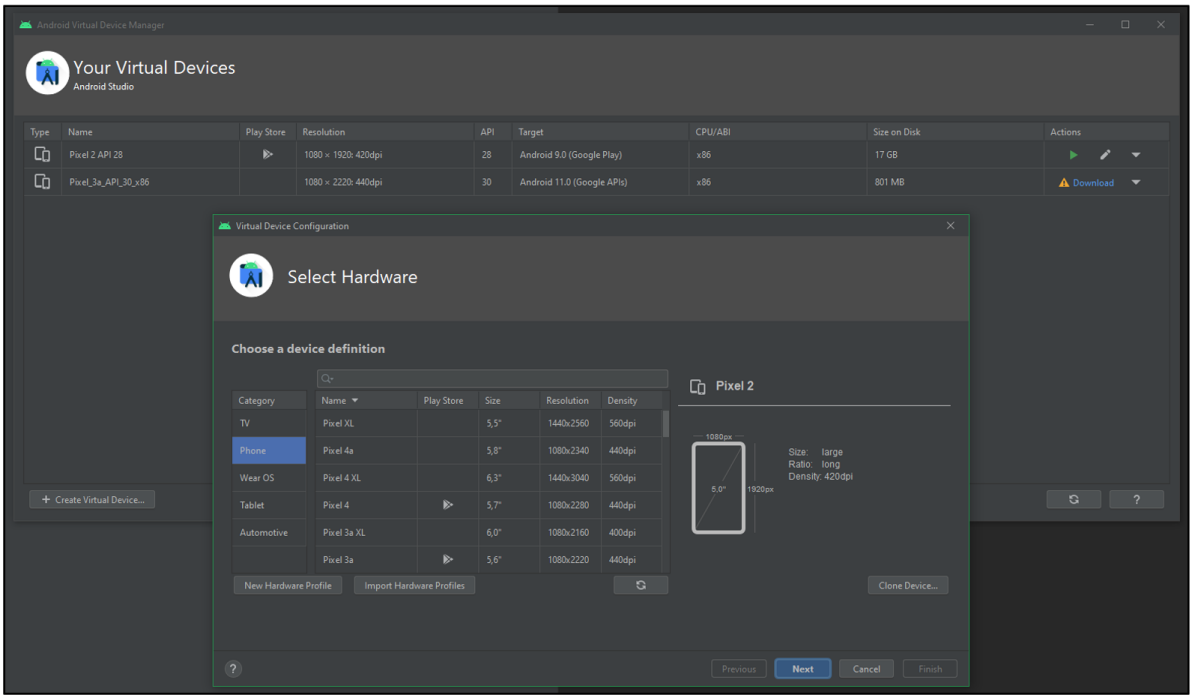
5.1.2. Fritzing

Fritzing je program korišten za izradu shema, tj. strujnih krugova, no na vizualno pristupačniji način. Primarna svrha softvera je upravo dokumentiranje Arduino prototipova te izrada PCB dizajna za proizvodnju. Program ne koriste isključivo inženjeri, nego dizajneri, umjetnici,

istraživači i hobisti prilikom razvoja projekata. Sam program je otvorenog tipa, napisan u C++ jeziku koji se može pronaći na GitHub-u. U idućem poglavlju se Fritzing koristi za izradu tzv. *breadboard* dijagrama svih napravljenih projekata. Dijagram se dobije na način da se elementi iz *toolboxa* povlače na kanvas te se međusobno spajaju linijama. Linije simboliziraju *jumper* žice kojima su povezani stvarni elementi. Na taj način se dobije pojednostavljeni model stvarnog projekta. Program ima mogućnost prebacivanja i na *circuit* način koji pruža detaljnu shemu strujnog kruga.

5.1.3. Android Studio

Kao pomoć pri programiranju mobilne aplikacije koristio se Android Studio, razvojno okruženje (eng. *Integrated Development Environment*) specijalizirano za razvoj Android mobilnih aplikacija. Razvijen od strane tvrtki Google i JetBrains, Android Studio kompatibilan je s macOS, Windows i Linux operacijskim sustavima. Iako se mobilna aplikacija u potpunosti mogla razviti u Android Studiu, odlučeno je koristiti samo jednu njegovu funkcionalnost, a ostatak je razvijen u Dartu/Flutteru. Ta funkcionalnost je AVD Manager (eng. *Android Virtual Device*). Zahvaljujući AVD-u moguće je kreirati, duplicirati, prilagođavati i pokretati Android virtualne uređaje. To znači da za razvoj i testiranje Android mobilne aplikacije nije neophodan fizički uređaj, samo virtualne instance koje pruža AVD.



Slika 16: Android Studio - Virtualni uređaji (Izvor: autorski rad)

U Android Studiu, na alatnoj traci, potrebno je odabrati opciju "Tools", a zatim "AVD Manager". Time se otvara novi prozor sa svim kreiranim virtualnim uređajima. Pokraj naziva uređaja nalaze se informacije o rezoluciji uređaja, verziji operacijskog sustava, vrsti procesora i količini memorijskog prostora koje zauzima na tvrdom disku. U donjem dijelu se nalazi gumb

"Create Virtual Device..." putem kojega se dodaju novi uređaji na popis. Kategorije uređaja su TV, mobitel, tablet, pametni satovi i aplikacije za automobile. Na popisu svih dodanih uređaja se pokraj svakog nalazi i zeleni start gumb, on pokreće virutalni uređaj. Nakon što se uređaj pokrene (npr. Android mobitel), pojavljuje se na ekranu te je moguće upravljati njime kao i s fizičkim uređajem. Naposljetku, moguće je zatvoriti i Android Studio i AVD Manager jer se više neće koristiti, važno je samo da je virtualni uređaj pokrenut. Razvoj nove aplikacije na tom uređaju se dalje nastavlja u programu Microsoft Visual Studio Code, koristeći Dart i Flutter.

5.1.4. ArduinoIDE - C/C++

Glavni element ovog rada su mikrokontroleri za čije programiranje je korišteno razvojno okruženje ArduinoIDE. Na prijašnjim stranicama ovog rada je već bilo govora o funkcionalnostima ArduinoIDE alata i načinu na koji se programiraju mikrokontroleri. Može se reći da je osnovna hardverska jedinica NodeMCU V3 razvojna ploča bazirana na ESP8266 SoC-u s integriranim WiFi modulom ESP-12E. NodeMCU je u suštini mikrokontroler implementiran na jednoj tiskanoj pločici, odnosno PCB-u. Zahvaljujući odličnom podržavanju Arduina od strane ESP8266, dovoljno je putem opcije *Board Manager* odabrati točnu razvojnu ploču koja će se koristiti nakon čega je sve spremno za programiranje. Ako se ploča ne nalazi na početnom popisu, jednostavno se preuzme na isti način kako se preuzimaju i biblioteke. Jezik koji se pritom koristi je C++ uz nekoliko metoda i funkcija specifičnih za ArduinoIDE i ovakav način programiranja.

Za svrhe ovog projekta izrađeno je pet različitih ".ino" skripti, po jedna za svaku NodeMCU razvojnu ploču. Skripte imaju neke zajedničke dijelove koda, poput spajanja na WiFi i povezivanja s Firebase-om, dok ostatak koda ovisi o namjeni projekta. Zato će prvo biti objašnjeni isječci koda prisutni u svim skriptama, a potom i ključni dijelovi koda specifičnih za svaku skriptu posebno. Sve skripte slijede određena pravila strukturiranja koda i raspored elemenata u kodu. Na početku se uvoze sve potrebne biblioteke, definiraju varijable i konstante, zatim se pišu sve funckije i metode, uključujući i dvije obavezne *setup* i *loop*.

5.1.4.1. Spajanje na WiFi

Temelj svake skripte je spajanje na WiFi mrežu. Kako bi to bilo izvedivo, potrebno je putem opcije *Library Manager* preuzeti odgovarajuću biblioteku. Ona u sebi sadržava sve procedure potrebne za spajanje razvojne ploče na internet. Nakon što se preuzme, biblioteka mora biti uključena u samu skriptu korištenjem dolje navedene naredbe.

```
//Pocetak skripte
#include <ESP8266WiFi.h>
```

Na isti način se dohvaćaju i sve ostale potrebne biblioteke, ovisno o ostalim povezanim modulima, senzorima i sl. Praksa je da se sve "include" naredbe nalaze na početku skripte. Nakon uvoza biblioteke potrebno je definirati dvije konstante koje će sadržavati dvije važne infromacije: naziv mreže i pripadajuću lozinku. Prilikom mijenjanja WiFi mreže i/ili lozinke dovoljno je samo promijeniti ova dva podatka.

```
//Pocetak skripte
#define WIFI_SSID "Naziv␣WiFi␣mreze"
#define WIFI_PASSWORD "Lozinka"
```

Prethodni podaci definiraju se na početku skripte, dok se samo spajanje na mrežu definira u *setup* metodi. Kako je već ranije ustanovljeno, *setup* metoda se izvrši samo jednom, na početku rada, a *loop* metoda se izvršava neprestano. U sljedećem isječku koda odvija se povezivanje na WiFi pomoću prethodno definiranih varijabli za naziv i lozinku mreže. Varijable se prosljeđuju kao parametri posebnoj metodi "WiFi.begin()". Proces spajanja detaljnije se može pratiti pokretanjem serijskog monitora unutar ArduinoIDE programa. Serijski monitor se koristi prilikom testiranja za ispis potrebnih informacija i logova naredbom "Serial.print()". Na taj način se direktno iz serijskog monitora može zaključiti je li spajanje na neku mrežu uspješno završeno. Nakon uspješnog povezivanja ispisuju se podaci o mreži i IP adresa spojenog uređaja. Time je postupak završen, a kod se idući put pokreće tek nakon resetiranja ploče.

```
//Setup metoda
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
Serial.print("Connecting␣to␣");
Serial.print(WIFI_SSID);
while (WiFi.status() != WL_CONNECTED)
{
    Serial.print(".");
    delay(500);
}
Serial.println();
Serial.print("Connected␣to␣");
Serial.println(WIFI_SSID);
Serial.print("IP␣Address␣is:␣");
Serial.println(WiFi.localIP());
WiFi.setAutoReconnect(true);
WiFi.persistent(true);
```

5.1.4.2. Spajanje na Firebase

Na sličan način kao kod spajanja na WiFi, odvija se i spajanje uređaja na Firebase. Firebase-om je omogućena komunikacija i prijenos podataka između uređaja i mobilne aplikacije. No, Kako bi to bilo ostvarivo, potrebno je preuzeti i uključiti potrebnu biblioteku u svaku skriptu dolje navedenom naredbom. Nakon seta "include" naredbi, u svakoj skripti se nalazi i skup informacija o projektu, definiranih kao konstante. One su izrazito bitne prilikom prvog pokretanja skripte i prvog spajanja na Firebase. Te informacije će se samo prvi put spremiti u bazu i neće se više spremati osim ako ih korisnik ne izmjeni. Nadalje, konstanta "path" se također definira u svakoj skripti i sadrži početni dio putanje do određenog projekta, ovisno o vrijednosti "PROJECT_INDEX" konstante. Svakom projektu dodijeljen je jedinstveni indeks (počevši od nula pa nadalje) čime je omogućeno jednostavnije iteriranje kroz projekte u samom kodu aplikacije.

```
//Pocetak skripte
#include <FirebaseArduino.h>
```

```
#define FIREBASE_HOST "naziv␣projekta"
#define FIREBASE_AUTH "autentikacijski␣kod"

const String PROJECT_NAME = "Plant␣station";
const String BOARD_NAME = "ESP8266␣ESP-12E";
const String CONNECTED_SENSORS = "DHT11␣+␣SoilMoisture";
const String BOARD_LOCATION = "Indoors";
const String PROJECT_INDEX = "0";
const String path = "/Stations/BrowseStations/" + PROJECT_INDEX;
```

Također, potrebno je definirati dvije varijable ključne za spajanje, a njih se može pronaći u Firebase postavkama nakon što se cijeli servis pripremi za korištenje. Kao i kod spajanja na WiFi, koristi se posebna naredba "Firebase.begin()" koja za svoje parametre prima prethodno definirane varijable, "Host" i "Auth". Cijeli proces povezivanja i eventualne pogreške se mogu ispisati u serijskom monitoru radi lakšeg pronalaska pogrešaka i testiranja. Nadalje, prethodno definirane konstante se u *Setup* metodi šalju na Firebase, na pripadajuće putanje, korištenjem naredbi "Firebase.setString()" i "Firebase.setBool()". Te metode za zapisivanje podataka na Firebase uvijek primaju dva argumenta, prvi je putanja do samog atributa, a drugi je vrijednost koju će taj atribut primiti.

```
//Setup metoda
Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);

Firebase.setString(path + "/BasicInfo/Name", PROJECT_NAME);
Firebase.setString(path + "/BasicInfo/Location", BOARD_LOCATION);
Firebase.setString(path + "/BasicInfo/Module", BOARD_NAME);
Firebase.setString(path + "/BasicInfo/Sensors", CONNECTED_SENSORS);
```

Naredbe slanja i dohvaćanja podataka će se kod svih projekata koristiti i u ostatku koda, u *Loop* metodama. Prva dva projekta se sastoje samo od senzora tako da će isključivo slati podatke navedenim "set" naredbama, ovisno o tipu podataka koji se šalje u bazu. Dok će ostali projekti osim slanja, morati i dohvaćati podatke korištenjem "get" naredbi, što će biti opisano kod projekta sa servo motorom. Razlika između "get" i "set" naredbi je i u broju parametara koje primaju. Naime, za "get" naredbu dovoljno je proslijediti samo putanju do željenog atributa, dok "set" naredba mora imati i proslijeđenu vrijednost kako bi ju mogla poslati u bazu.

```
//Firebase connection
    void firebasereconnect(){
    delay(300);
    Serial.println("Trying␣to␣reconnect");
    Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
    delay(1000);
}

//Loop metoda
//Firebase reconnect
  if (Firebase.failed()){
    Serial.print("setting␣number␣failed:");
    Serial.println(Firebase.error());
    firebasereconnect();
```

```
    return;
}
```

Obično se između dviju obaveznih metoda definiraju ostale, pomoćne metode. Tako se u ovom slučaju definira metoda "firebasereconnect()" koja se poziva samo u slučaju prekida veze između uređaja i Firebase-a. Pozivat će se odmah na početku *loop* metode, gdje se na početku svake iteracije provjeri veza s Firebase-om. Na taj način je spriječeno nepotrebno izvođenje ostatka programa, dok god se ponovno ne uspostavi veza s bazom.

5.1.4.3. Dohvaćanje vremenske oznake

Dohvaćanje vremenske oznake posljednji je isječak koda prisutan u svim skriptama. Njegova uloga je jednostavna, a to je da putem interneta dohvaća trenutnu vremensku oznaku i šalje ju svakih nekoliko sekundi na Firebase bazu podataka. U mobilnoj aplikaciji postoji dio koda zadužen za dohvaćanje tih vremenskih oznaka i uspoređivanje s trenutnim vremenom. Kada se uređaj ugasi, prestaje slanje vremenskih oznaka, a u bazi ostaje posljednja poslana oznaka. Na taj način će u aplikaciji u jednom trenutku posljednja vremenska oznaka previše odskakati od stvarnog vremena. Tada aplikacija može promijeniti status uređaja iz statusa "Connected" u "Not connected". Dakle, korištenje vremenske oznake u skriptama služi samo za kontrolu aktivnosti uređaja i prikaz statusa u aplikaciji.

Kao i do sada, osnovni korak je uključivanje potrebnih biblioteka i definiranje varijabli u početnom dijelu skripte.

```
//Pocetak skripte
#include <NTPClient.h>
#include <WiFiUdp.h>

WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP);
```

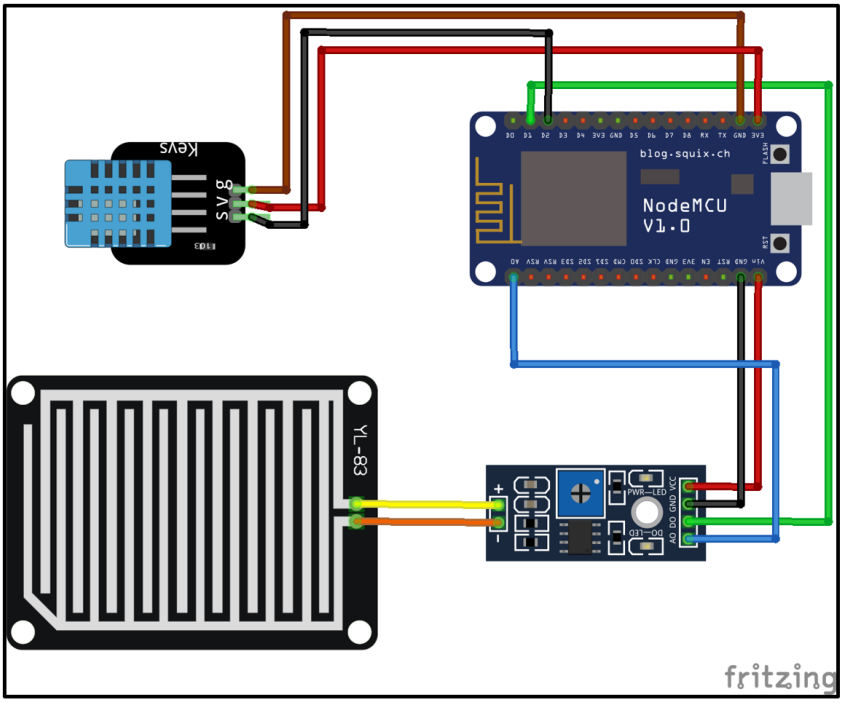
U *setup* metodi se naredbom "timeClient.begin()" pokreće klijent za dohvaćanje trenutne vremenske oznake.

```
//Setup metoda
timeClient.begin();
```

U *loop* metodi se definira dio koda koji dohvaća vrijeme. Vremenska oznaka se dohvaća u tzv. *Unix epoch* obliku. To je ukupan broj proteklih sekundi od 01. siječnja 1970. godine. Iako dohvaćeni broj izgleda prilično nezgrapno, odlično je rješenje za jednostavno uspoređivanje vremenskih oznaka. Oznaka se zbog lakšeg rukovanja konvertira u "String" tip podataka i sprema u varijablu, nakon čega je spremna za slanje na Firebase.

```
//Loop metoda
timeClient.update();
String fireTime = String(timeClient.getEpochTime());
```

5.1.4.4. Vremenska stanica



Slika 17: Dijagram vremenske stanice (Izvor: autorski rad)

Vremenska stanica sastavljena je od NodeMCU V3 razvojne ploče na koju su povezani senzori za temperaturu i vlagu (DHT11) i senzor za kišu, DHT11 modul ima tri pina koja su povezana spomenutim *jumper* žicama i to na način da se odgovarajuće pinove spoji s "GND" pinom za uzemljenje, "3v" pinom za napajanje DHT11 modula i "D3" pinom za slanje digitalnog signala. Senzor za kišu također je spojen na "GND" i "3V" pinove iz istih razloga te na "A0" analogni pin za slanje informacija o jačini padalina. Stanica s navedenim senzorima se postavi na otvorenom i redovno šalje podatke o temperaturi, vlažnosti zraka i informacije o tome pada li kiša i koliko jako.

```
//Pocetak skripte
#include <DHT.h>

#define DHTPIN D2
#define DHTTYPE DHT11
#define rainAnalog A0
#define rainDigital D1
DHT dht(DHTPIN, DHTTYPE);
```

U početnom dijelu skripte se standardno uvozi biblioteka za DHT11 senzor i definiraju se varijable ovisno o pinovima na koje su senzori spojeni. Iz dijagrama se vidi da je DHT senzor spojen na digitalni pin "D2", a senzor za kišu spojen je na digitalni pin "D1" i na analogni pin "A0". Osim toga, kreira se i "dht" objekt tako da mu se proslijede prethodno definirani podaci o pinu na koji je spojen i tipu DHT senzora, u ovom slučaju DHT11.

```
//Setup metoda
```

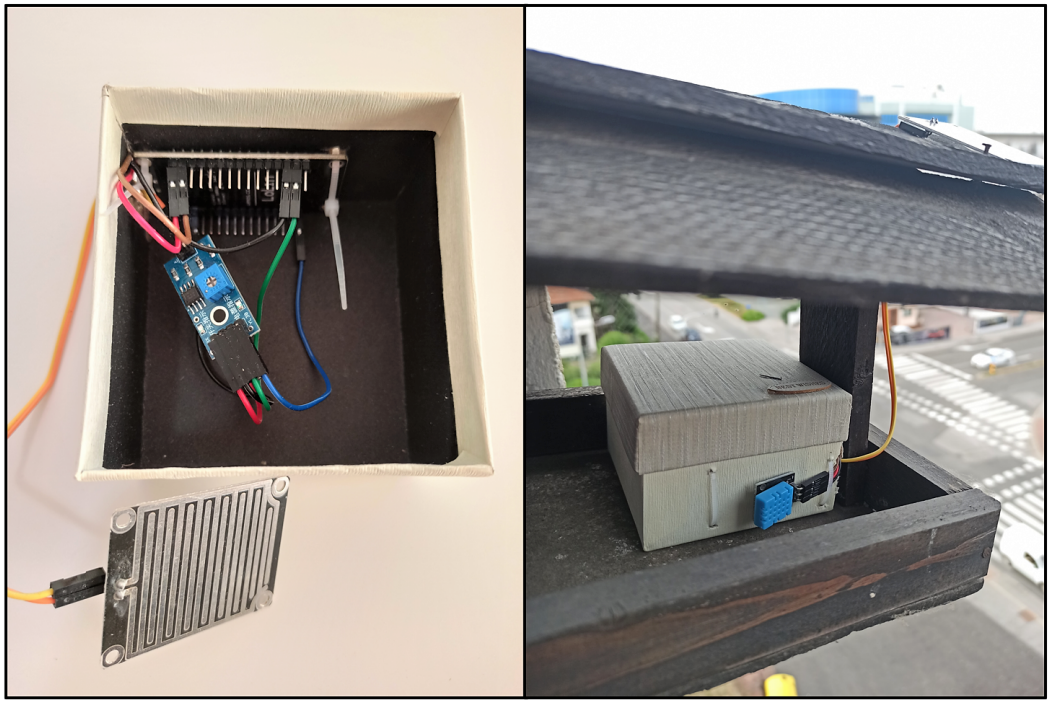
```
dht.begin();
pinMode(rainDigital,INPUT);
```

U *setup* funkciji senzori se inicijaliziraju tako da se pozove "begin()" funkcija pomoću "dht" objekta, a za senzor za kišu je dovoljno definirati pin na koji će stizati ulazni podaci od senzora. U donjem isječku koda se "dht" objektom pozivaju metode "readHumidity()" i "readTemperature()", a dobivene vrijednosti se spremaju u korespondirajuće varijable. Nakon dohvaćanja podataka odvija se provjera ispravnosti i pretvaranje vrijednosti u oblik pogodan za slanje na Firebase.

```
//Loop metoda
float h = dht.readHumidity();
float t = dht.readTemperature();
if (isnan(h) || isnan(t)) {
    Serial.println(F("Failed␣to␣read␣from␣DHT␣sensor!"));
    return;
}
String fireHumid = String(h);
String fireTemp = String(t);
```

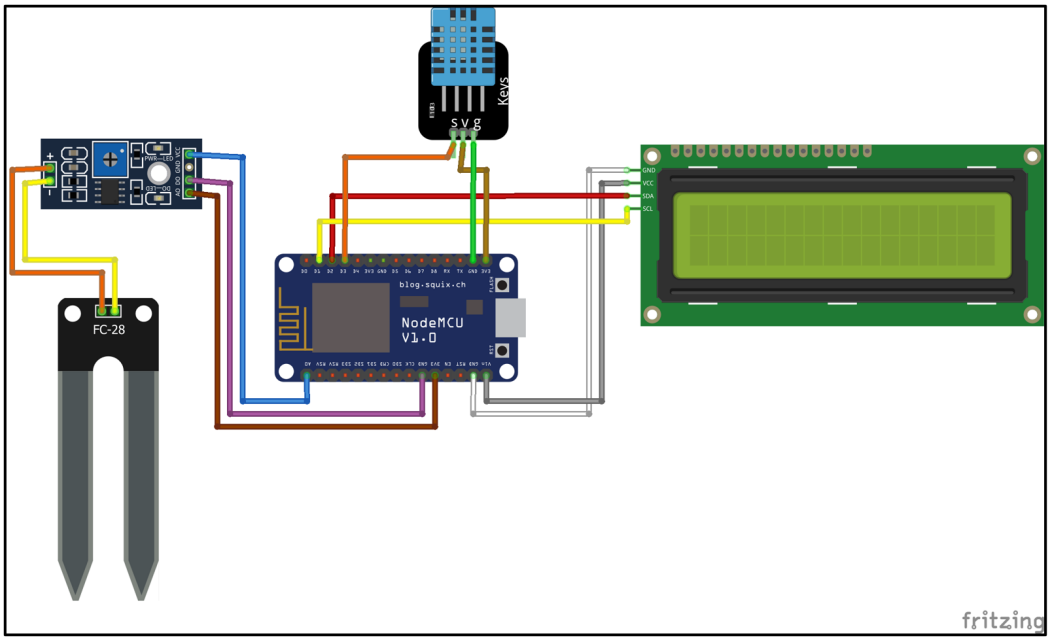
Najbitniji dio koji se tiče senzora za kišu je dohvaćanje podataka s analognog pina pomoću "analogRead()" funkcije. Dohvaćena analogna vrijednost će biti broj između 0 i 1023, gdje manji broj označava jači intenzitet padalina. Zato se kroz niz "if-else" blokova uspoređuje analogna vrijednost s graničnim vrijednostima na temelju čega se određuje intenzitet. Osim što se ispiše, intenzitet se sprema i u varijablu kako bi se vrijednost kasnije slala na Firebase.

```
//Loop metoda
int rainAnalogVal = analogRead(rainAnalog);
int rainDigitalVal = digitalRead(rainDigital);
String fireRainAnalog;
if (rainAnalogVal > 1015) {
    fireRainAnalog = "No␣rain!";
    Serial.println(fireRainAnalog);
}
else if (rainAnalogVal > 850 && rainAnalogVal <= 1015) {
    fireRainAnalog = "Light␣rain!";
    Serial.println(fireRainAnalog);
}
else if (rainAnalogVal > 400 && rainAnalogVal <= 850){
    fireRainAnalog = "Medium␣rain!";
    Serial.println(fireRainAnalog);
}
else {
    fireRainAnalog = "Heavy␣rain!";
    Serial.println(fireRainAnalog);
}
```



Slika 18: Vremenska stanica (Izvor: autorski rad)

5.1.4.5. Stanica za biljke



Slika 19: Dijagram biljne stanice (Izvor: autorski rad)

Stanica za biljke je također koncipirana tako da očitava i obrađuje podatke iz okoline. Budući da je prikladnija za sobnu upotrebu, ima spojen i LCD ekran za ispis podataka: temperature, vlage i vlažnosti tla. Za temperaturu i vlagu se i ovdje koristi DHT11 senzor, a za vlažnost tla se koristi *Soil Moisture Sensor* koji se zabode u zemlju.

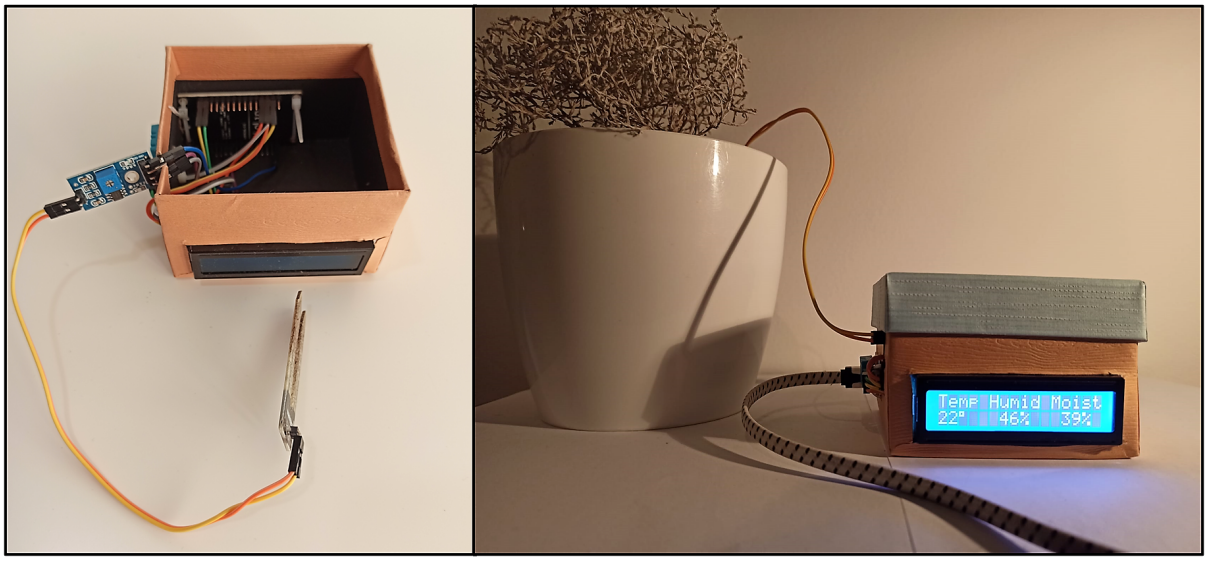
```
//Pocetak skripte
#include <LiquidCrystal_I2C.h>
#include <DHT.h>

#define DHTPIN D3
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
const int sensor_pin = A0;
LiquidCrystal_I2C lcd(0x27, 16, 2);

//Setup metoda
dht.begin();
lcd.begin(16,2);
lcd.init();
lcd.backlight();
```

Kako bi se sve elemente moglo ispravno koristiti prvo se uvoze potrebne biblioteke za DHT11 i za LCD ekran. Varijable vezane uz DHT su objašnjene na prethodnom primjeru, definira se varijabla za vlažnost na "A0" analognom pinu i poseban objekt za LCD ekran. Nakon toga se u *setup* dijelu inicijalizira DHT11 senzor i LCD standardnim "begin()" metodama, uz napomenu da se kod LCD-a prosljeđuju parametri širine i visine ekrana. Ekran je napravljen u obliku matrice 16x2 gdje u svako polje može stati jedan znak. Naredbama "lcd.init()" i "lcd.backlight()" dovršava se potupak inicijalizacije LCD ekrana.

```
//Loop metoda
float moisture_percentage;
moisture_percentage = (100.00-((analogRead(sensor_pin)/1023.00)*100.00));
String fireMoist = String(moisture_percentage);
String lcdMoist = String((int)round(moisture_percentage)) + String("%");
delay(500);

lcd.setCursor(0, 0);
lcd.print("Temp");
lcd.setCursor(5, 0);
lcd.print("Humid");
lcd.setCursor(11, 0);
lcd.print("Moist");
lcd.setCursor(0, 1);
lcd.print(lcdTemp);
lcd.setCursor(2, 1);
lcd.print((char)223);
lcd.setCursor(6, 1);
lcd.print(lcdHumid);
lcd.setCursor(12, 1);
lcd.print(lcdMoist);
delay(5000);
```

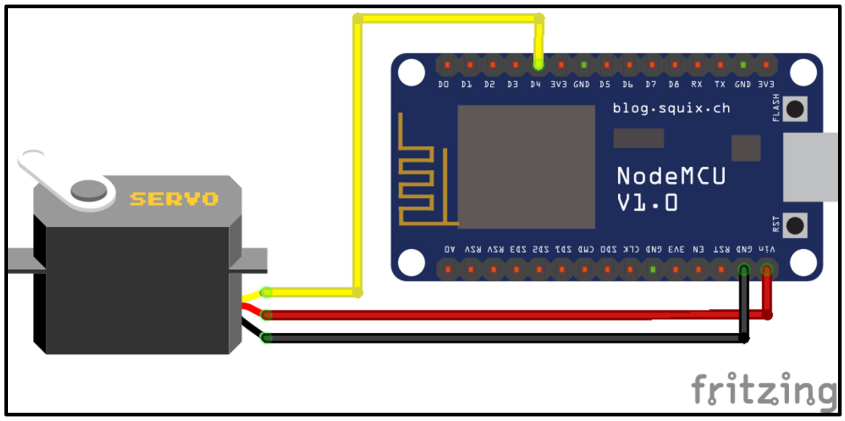
U *loop* dijelu se pomoću određene formule dobiva vlažnost tla u obliku postotka. Odmah se pretvara u string tip podatka za slanje na Firebase, a vrijednost se obrađuje i za pogodan prikaz na ekranu. Zaokružuje se na najbliži cijeli broj te se takvom stringu dodaje i simbol za postotak. Za ispis na ekran potrebno je prije svakog ispisa nekog stringa postaviti "kursor" na

željenu poziciju matrice naredbom "lcd.setCursor()". Zatim se redom postavlja kursor po matrici i ispisuju svi stringovi.



Slika 20: Stanica za biljke (Izvor: autorski rad)

5.1.4.6. Hranilica za ljubimce



Slika 21: Dijagram hranilice za ljubimce (Izvor: autorski rad)

Ovaj projekt se sastoji samo od jedne dodatne komponente, a to je mali servo motor. On će se upaliti kada skripta iz baze dohvati određen podatak, točnije kada dohvati *boolean* vrijednost postavljenu na "true". Ovo je prvi primjer u kojem je potrebno dohvaćati podatke s Firebasea, a ne samo spremati. Na rotacijski dio servo motora pričvršćena je pločica koja ima ulogu vrata. Kada se motor aktivira, vrata se otvaraju, a kroz vrata cure sjemenke za zamorce.

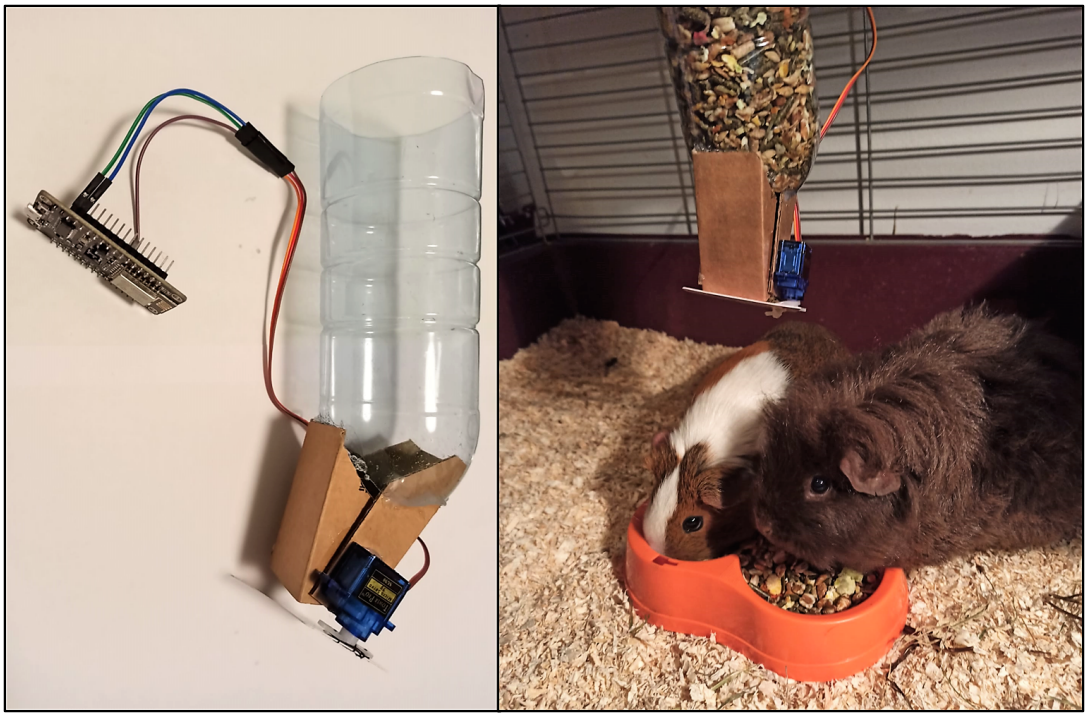
```
//Pocetak skripte
#include <Servo.h>

Servo myservo;
int pos = 0;
bool switcher = false;
```

```
//Setup metoda
myservo.attach(2);
```

Nakon uvoza biblioteke za servo motor i kreiranja objekta za upravljanje motorom, u *setup* segmentu dovoljno je samo pridružiti objektu vrijednost pina na koji je spojen. Na dijagramu je vidljivo da je motor spojen na pin "D4", dok proslijeđeni parametar u kodu iznad ima vrijednost 2. To je zato što svaki pin na NodeMCU razvojnoj ploči ima posebnu pridruženu vrijednost u ArduinoIDE okruženju. U ovom slučaju, korespondirajuća vrijednost pina "D4" u ArduinoIDE je 2. Tablica svih pinova s pridruženim ArduinoIDE vrijednostima i aliasima napravljena je u prvom dijelu rada, u odlomku o ESP8266.

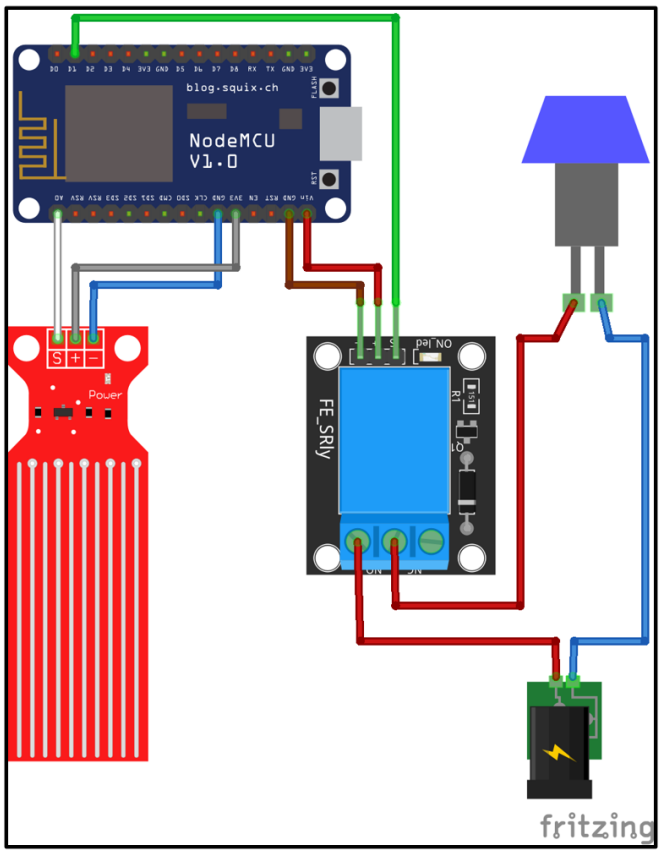
```
//Loop metoda
switcher=Firebase.getBool(path + "/Interactive/Servo");
if(switcher==false){
    delay(1000);
}
if(switcher==true){
    for (pos = 90; pos <= 270; pos += 1) {
        myservo.write(pos);
        delay(1);
    }
    for (pos = 270; pos >= 90; pos -= 1) {
        myservo.write(pos);
        delay(1);
    }
delay(2000);
switcher = false;
Firebase.setBool(path + "/Interactive/Servo", switcher);
delay(1500);
}
```

Skripta dohvaća vrijednost "Servo" atributa iz baze podataka i sprema u varijablu "switcher". Dohvaćanje se postiže naredbom "Firebase.getBool()" koja za parametar prima putanju do željenog atributa. Ako je vrijednost *false*, znači da se servo ne treba paliti tako da se u tom slučaju ništa ni ne događa osim mirovanja od jedne sekunde. Ako skripta dohvati *true* vrijednost servo se pokreće pomoću dvije "for-petlje". Naredbom "myservo.write()" se postiže "pisanje" na servo, na način da će se rotacijski dio motora pomaknuti na poziciju, odnosno kut, koji se prosljedi kao parametar. U ovom slučaju se servo kroz dvije petlje rotira za 180 stupnjeva u jednom smjeru, a nakon toga odmah i u drugom smjeru. Time se dobio efekt brzog otvaranja i zatvaranja vrata. Testiranjem je utvrđeno da ovaj način postiže ispuštanje optimalne količine sjemenki. Po završetku petlji vrijednost atributa u bazi se vraća na vrijednost *false* metodom "setBool()". Metoda prima dva parametra: prvi je putanja do traženog atributa, a drugi je vrijednost koju će taj atribut poprimiti, tj. *false*.



Slika 22: Hranilica za ljubimce (Izvor: autorski rad)

5.1.4.7. Relej sklopka



Slika 23: Dijagram relej sklopke (Izvor: autorski rad)

Relej sklopka, ili *Relay Switch*, funkcionira na principu običnog prekidača za svjetlo. Njegova namjena je otvaranje ili zatvaranje strujnog kruga što se postiže slanjem signala s mikrokontrolera. Pomoću NodeMCU razvojne ploče to je moguće činiti i na daljinu. U stvarnom projektu prekidač je ugrađen u produžni kabel pa se tako bilo koji kućanski aparat i uređaj može paliti i gasiti preko mobilne aplikacije. Osim toga, dodan je i senzor za očitavanje razine vode postavljen u akvarij s vodom. Naime, u produžni kabel je uštekan obični akvarijski filter s cijevi na vrhu iz koje vraća vodu u akvarij. On je prenamijenjen tako da voda ne curi natrag u akvarij nego u lončanicu sa zemljom. Time se dobio jednostavan sustav za navodnjavanje koji u isto vrijeme može pokretati i zaustavljati navodnjavanje zemlje te pratiti trenutnu razinu vode u akvariju.

```
//Pocetak skripte
int waterValue = 0;
String waterLevel;
int waterPin = A0;
int relay = D1;
bool switcher = false;
```

Definiranje varijabli se odvija kao i inače, senzor za razinu vode spojen je na analogni "A0" pin, a prekidač na digitalni "D1" pin.

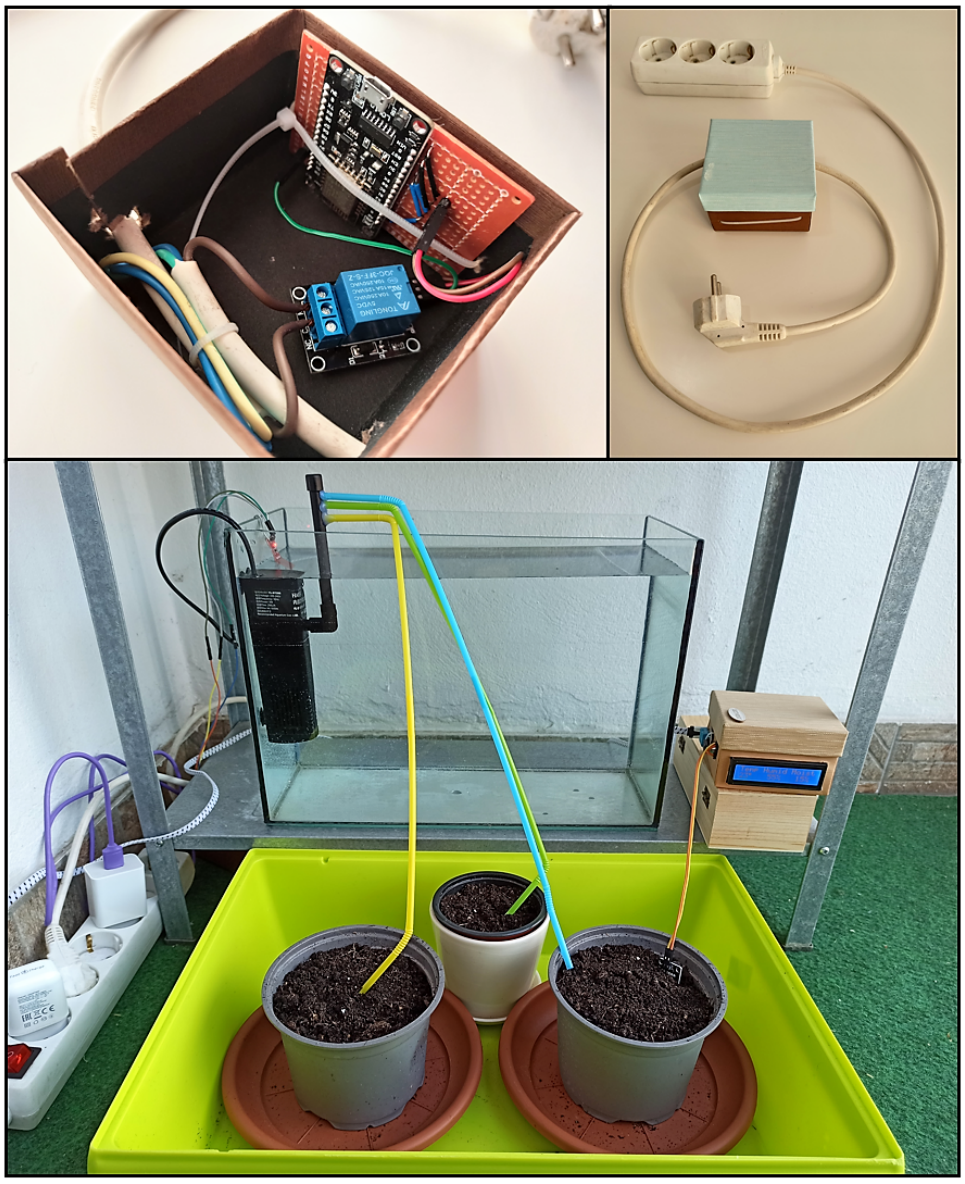
```
//Loop metoda
waterValue = analogRead(waterPin);
if (waterValue <= 150) {
    waterLevel = "Empty";
}
else if (waterValue > 150 && waterValue <= 250) {
    waterLevel = "Very Low";
}
else if (waterValue > 250 && waterValue <= 350) {
    waterLevel = "Low";
}
else if (waterValue > 350 && waterValue <= 500) {
    waterLevel = "Medium";
}
else if (waterValue > 500 && waterValue <= 600) {
    waterLevel = "High";
}
else if (waterValue > 600) {
    waterLevel = "Full";
}
```

Naredbom "analogRead()" skripta iščitava vrijednost s "A0" pina i sprema u varijablu "waterValue". Vrijednosti će biti u rasponu 0-1023, kao i ostali analogni senzori. Ta vrijednost se provlači kroz niz "if-else" blokova na temelju čega se određuje kolika je razina vode u akvariju. Kada vrijednost pronađe interval kojem pripada, varijabla "waterLevel" poprima vrijednost razumljivu korisniku koja se zatim šalje u bazu i prikazuje u aplikaciji.

```
//Loop metoda
switcher=Firebase.getBool(path + "/Interactive/Relay");
if(switcher==false) {
```

```
        digitalWrite(relay,LOW);
    }
    if(switcher==true) {
        digitalWrite(relay,HIGH);
    }
```

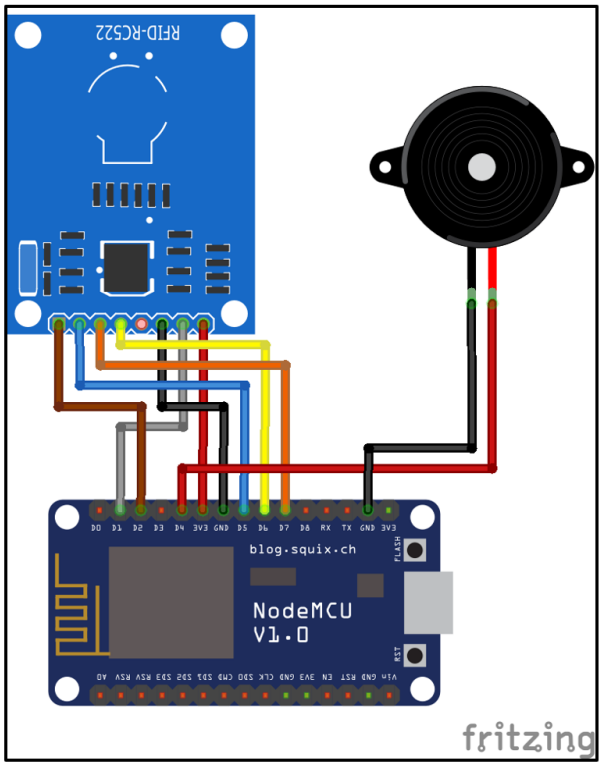
Relej funkcionira na sličan način kao i prethodni servo motor. Naredbom "getBool()" dohvaća vrijednost "Relay" atributa iz baze te određuje svoje ponašanje na temelju vrijednosti. Ona će biti *false* dok god korisnik ne pomakne klizni gumb u aplikaciji na "ON". U tom trenutku se vrijednost u bazi mijenja u *true*, skripta to pročita i šalje naredbu releju da se upali. Razlika između servo motora je što se u aplikaciji servo motor pali običnim gumbom, odnosno vrijednost *true* stoji u bazi samo dok se jednom ne izvrši servo akcija. U ovom slučaju, vrijednsot ostaje *true* dok god korisnik u aplikaciji ne vrati klizni gumb na vrijednost "OFF". Tako korisnik može kontrolirati vrijeme navodnjavanja po želji.



Slika 24: Relej sklopka (Izvor: autorski rad)

Kako je vidljivo na slici 24, relej sklopci je u praksi dodana i stanica za biljke. Time je kreiran kompletan sustav koji može navodnjavati zemlju vodom iz spremnika, pratiti razinu vode u spremniku te izvještavati o vlažnosti zemlje u lončanici, vlažnosti i temperaturi zraka.

5.1.4.8. RFID čitač



Slika 25: Dijagram RFID čitača (Izvor: autorski rad)

Ovaj projekt sastoji se od NodeMCU razvojne ploče, RFID čitača i malog zvučnika, *Mini Piezo Buzzer*. Za aktiviranje čitača koristi se RFID kartica i RFID privjesak za ključeve. Oba predmeta imaju jedinstveni "otisak" koji se pomoću skripte dohvati i ispiše na serijski monitor. Budući da je otisak predmeta stalan i jedinstven, svi otisci se kopiraju sa serijskog monitora i spreme za kasnije korištenje. Zvučnik je dodan zbog praktičnosti, ali i zbog efikasnijeg testiranja.

```
//Pocetak skripte
#include <SPI.h>
#include <MFRC522.h>

constexpr uint8_t RST_PIN = 5;
constexpr uint8_t SS_PIN = 4;
MFRC522 rfid(SS_PIN, RST_PIN);
MFRC522::MIFARE_Key key;
byte nuidPICC[4];

//Setup metoda
SPI.begin();
```

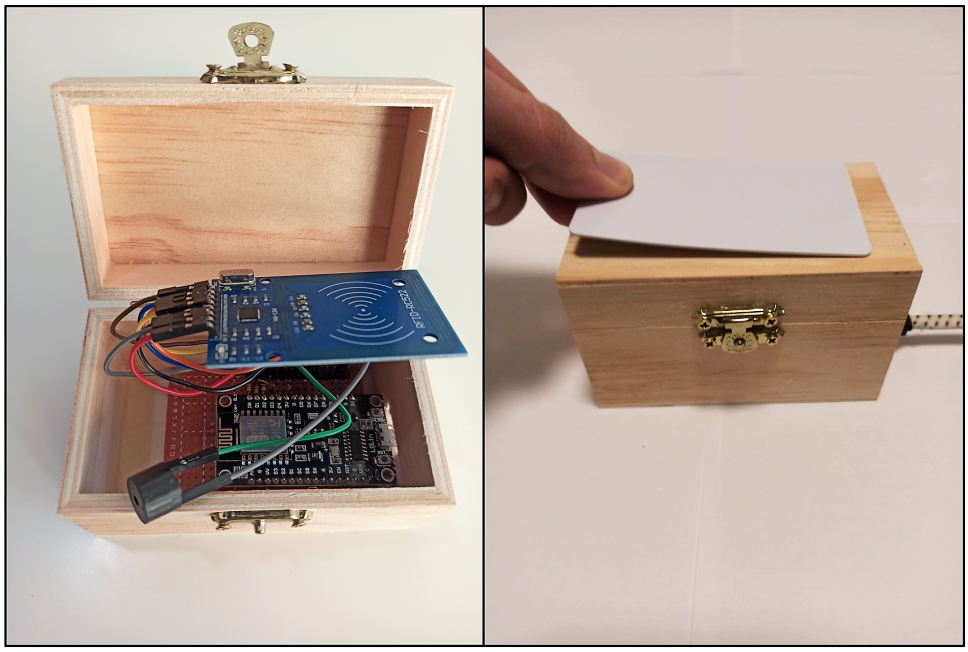
```
rfid.PCD_Init();
```

"SPI.h" biblioteka omogućava brzu, sinkronu komunikaciju između mikrokontrolera i perifernih uređaja. "MFRC522.h" biblioteka služi za upravljanje RFID čitačem i očitavanje prisutnosti RFID predmeta. Varijable se pridružuju prema pinovima na koje su spojene, kreiraju se potrebni objekti, a u *setup* metodi se objekti inicijaliziraju.

```
//Loop metoda
if ( ! rfid.PICC_IsNewCardPresent())
    return;
if ( ! rfid.PICC_ReadCardSerial())
    return;
MFRC522::PICC_Type piccType = rfid.PICC_GetType(rfid.uid.sak);
String content="";
for (byte i = 0; i < rfid.uid.size; i++) {
    content.concat(String(rfid.uid.uidByte[i] < 0x10 ? "␣0" : "␣"));
    content.concat(String(rfid.uid.uidByte[i], HEX));
}
content.toUpperCase();
if (content.substring(1) == "69␣C5␣C9␣55"){
    Serial.println("This␣is␣the␣card␣tag");
    String countString = Firebase.getString(path + "/Counter/Counter_1");
    count1 = countString.toInt();
    count1 = count1+1;
    Firebase.setString(path + "/Counter/Counter_1", String(count1));
    tone(buzzer,500);
    delay(200);
    noTone(buzzer);
}
else {
    Serial.println("This␣is␣the␣keychain␣tag");
    String countString = Firebase.getString(path + "/Counter/Counter_2");
    count2 = countString.toInt();
    count2 = count2+1;
    Firebase.setString(path + "/Counter/Counter_2", String(count2));
    tone(buzzer,250);
    delay(100);
    noTone(buzzer);
}
rfid.PICC_HaltA();
rfid.PCD_StopCrypto1();
```

U nastavku se posebnim metodama provjerava prisutnost RFID objekta u blizini RFID čitača. Ako se objekt detektira, u "content" varijablu se sprema njegov "otisak". Buduće ponašanje skripte se tada može uvjetovati tim "otiskom". Ovisno o tome koliko RFID objekata korisnik posjeduje, može definirati isto toliko "if" blokova, po jedan za svaki predmet posebno. U ovom slučaju se koriste samo dva objekta pa je dovoljno provjeriti samo vrijednost prvog u "if" uvjetu, a "else" dio će se izvršiti kod detekcije drugog objekta. Kod unutar "if" i "else" blokova u suštini radi istu radnju, a to je dohvaćanje vrijednosti iz baze, ažuriranje vrijednosti te slanje nove vrijednosti u bazu. U bazi podataka se nalaze dva atributa brojača, po jedna za svaki RFID objekt, što omogućava dva odvojena brojača u aplikaciji. Brojači mogu imati

drugačije namjene, a moguće je imati i brojače istih namjena, ali za dva korisnika. Naredbama "tone()" i noTone()" upravlja se malim zvučnicima koji proizvode zvuk nakon detekcije objekta. Kombinacijom "tone()" i delay()" naredbi moguće je i "odsvirati" razne melodije. Upravo to je i učinjeno u pravoj skripti, no u ovom isječku koda taj dio je skraćen zbog lakše preglednosti.



Slika 26: RFID čitač (Izvor: autorski rad)

5.1.5. Microsoft Visual Studio Code - Dart i Flutter

Visual Studio Code pripada kategoriji uređivača izvornog koda (eng. *Source code editor*), za razliku od Visual Studia ili Android Studia koji su razvojna okruženja. Postavljanje Flutter frameworka na Visual Studio Code je jednostavno, potrebno je samo odabrati opciju "Extensions", pronaći Flutter i instalirati. Instaliranjem Fluttera se pristupa svim funkcionalnostima koje nudi, a pripadnim SDK-om (eng. *Software Development Kit*) moguće je, u konačnici, kompilirati napisani kod. Uz Flutter će se automatski instalirati i Dart ekstenzija.

Dart je programski jezik čiji glavni fokus je na razvoju *front-end* korisničkih sučelja i aplikacija. Neovisan je o Flutteru te se njime mogu razvijati i web aplikacije. U ovom projektu naglasak nije na samom Dartu nego na tome kako ga Flutter koristi za razvoj mobilnih aplikacija. Flutter je *framework* za Dart jezik, odnosno kolekcija alata, funkcija i widgeta koji se implementiraju pomoću Dart jezika. Cilj Flutter frameworka je učiniti razvoj mobilnih višeplatformskih aplikacija što jednostavnijim. Ključna riječ je višeplatformski (eng. *cross-platform*), što označava mogućnost generiranja aplikacija za različite platforme na temelju samo kodne baze. Kod napisan u Dartu/Flutteru se pomoću Flutter SDK kompilira u izvorni kod (eng. *native code*) posebno za Android i za iOS. Rezultat toga su optimizirane aplikacije visokih performansi.

Programiranje mobilnih aplikacija u Flutteru ne podrazumijeva korištenje vizualnog editora s *drag-and-drop* elementima. Korisničko sučelje se gradi isključivo kodiranjem i konstruiranjem tzv. stabla widgeta. Widget je osnovni građevni element u Flutteru. Cijela aplikacija zapravo jedan kompleksni widget (najčešće "MaterialApp") koji za argumente prima druge widgete i tako dalje. Flutter dolazi s gomilom postojećih widgeta, ali moguće je kreirati i vlastite ako je to potrebno. Svaki zaslon kojeg korisnik vidi će u suštini biti klasa koja proširuje jedan od dva *state* widgeta: "StatefulWidget" ili "StatelessWidget". "StatelessWidget" je uobičajen widget koji služi za jednostavan prikaz podataka i statičnog korisničkog sučelja. Međutim, ako se podaci moraju osvježavati i mijenjati u stvarnom vremenu, potrebno je koristiti "StatefulWidget" koji ima mogućnost ažuriranja korisničkog sučelja.

Slika 27: Aplikacija - *Splash screen* (Izvor: autorski rad)

Prilikom pokretanja aplikacije se umjesto uobičajenog zaslona učitavanja (eng. *loading screen*) prikazuje tzv. *splash* zaslon, prikazan na slici 27. *Splash* zasloni se mogu konfigurirati posebno za Android i iOS uređaje. U projektu je potrebno urediti datoteku "launch_background.xml" koja se nalazi u "android" direktoriju. Modificiranjem datoteke korisnik može prilagoditi *splash* zaslon po želji. U slučaju dodavanja vlastite slike, potrebno ju je prvo dodati u "mipmap" folder.

Projekt se sastoji od ukupno šest ".dart" datoteka koje se nalaze u "lib" direktoriju. Datoteka "main.dart" sadržava osnovnu funkciju "main()" koja se automatski prva izvršava i pokreće cijelu aplikaciju naredbom "runApp()". Toj funkciji se prosljeđuje klasa koja proširuje spomenuti "StatelessWidget" i vraća temeljni widget "MaterialApp". U njemu se definira tema koja će se prožimati kroz cijelu aplikaciju. Tema podrazumijeva boju pozadine, boju aplikacijske trake, font teksta, primarnu lepezu boja i slično. Obavezan parametar "home" je pokazivač na prvi zaslon koji će se prikazati korisniku.

```
void main() {
  runApp(MyApp());
}
```

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'WoTsApp',
      theme: ThemeData(
        scaffoldBackgroundColor: Color(0xFFE1DFDF),
        textTheme:
            GoogleFonts.cabinCondensedTextTheme(Theme.of(context).textTheme),
        primarySwatch: Colors.blue,
        primaryColor: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
      home: ScreenStart(),
    );
  }
}
```

Klasa "ModulesCallFB.dart" je pomoćna datoteka u kojoj se odvija komunikacija s Firebaseom. Program šalje zahtjev za snimkom baze podataka, a kao odgovor prima JSON datoteku.

```
    DataSnapshot dataSnapshot = await databaseReference.once();
    Map<dynamic, dynamic> jsonResponse = dataSnapshot.value['Stations'];
```

JSON odgovor se prosljeđuje i obrađuje u datoteci ModulesList.dart. Ondje se odvija iteracija kroz JSON strukturu i razdvajanje svih modula u pomoćnu listu. Iteriranju pomažu i indeksi definirani u svakoj Arduino skripti koji se koriste pri konstruiranju Firebase stabla. Potrebno je istaknuti da se jedan modul odnosi na jedan Arduino projekt. Svaki se modul zatim posebno raščlanjuje prema ključnim riječima (npr. *Numerical, NonNumerical, Interactive*, itd.) te se podaci pridružuju pripadajućoj mapi. U konačnici se dobiva lista instanci klase ModuleDetails koja je sačinjena od određenog broja mapa. Svaka mapa sadržava određene segmente informacija o projektu, ovisno o tome kako je projektirano stablo u Firebase bazi podataka. Mapa je kolekcija parova ključeva i vrijednosti, a u ovim mapama su svi elementi definirani kao dinamički tip podataka. Na taj način je postignuta određena razina fleksibilnosti i mogućnost da tip podataka spremljenih u bazu može varirati.

```
class ModuleDetails {
  Map<dynamic, dynamic> numericalReadings;
  Map<dynamic, dynamic> nonNumericalReadings;
  Map<dynamic, dynamic> interactive;
  Map<dynamic, dynamic> basicInfo;
  Map<dynamic, dynamic> counter;
  Map<dynamic, dynamic> counterNames;
  Map<dynamic, dynamic> timestamp;

  ModuleDetails(
     {this.numericalReadings,
      this.nonNumericalReadings,
      this.interactive,
      this.basicInfo,
```

```
      this.counter,
      this.counterNames,
      this.timestamp});

  factory ModuleDetails.fromJson(Map<dynamic, dynamic> parsedJson) {
    return ModuleDetails(
        numericalReadings: parsedJson['Numerical'],
        nonNumericalReadings: parsedJson['NonNumerical'],
        interactive: parsedJson['Interactive'],
        basicInfo: parsedJson['BasicInfo'],
        counter: parsedJson['Counter'],
        counterNames: parsedJson['CounterNames'],
        timestamp: parsedJson['Time']);
  }
}
```

Time je završeno opisivanje dijela aplikacije odgovornog za dohvaćanje i obradu podataka iz baze. Preostaje opisati ostatak koda koji će te podatke pregledno prikazati na zaslon. Aplikacija se sastoji od dvaju zaslona: od početnog zaslona na kojem su prikazani svi povezani uređaji i *dashboarda* za svaki projekt.

Početni zaslon je klasa koja proširuje "StatefulWidget" zato što se temeljni widget za vrijeme trajanja svog životnog vijeka može naknadno mijenjati. Aplikacija nema fiksne i unaprijed poznate podatke, nego izgled početnog zaslona ovisi o podacima koji se nalaze u bazi podataka. Zbog toga se koristi poseban "FutureBuilder" widget. Naime, on samog sebe izgrađuje na temelju posljednjeg snimka iz baze. Zaslon će se zato uvijek automatski ažurirati ako se dogode promjene u bazi podataka.

Glavna klasa prvog zaslona kao odgovor vraća "Scaffold" widget koji uvelike olakšava posao dizajniranja izgleda aplikacije. "Scaffold" sadržava sve potrebne elemente za ostvarivanje izgleda jednostavne aplikacije. Uključuje aplikacijsku traku na vrhu zaslona na koju se može dodati naslov, definiranje glavnog sadržaja aplikacije, navigacijsku traku na dnu zaslona, plutajuće gumbe, pozadinu zaslona i ostale funkcije.

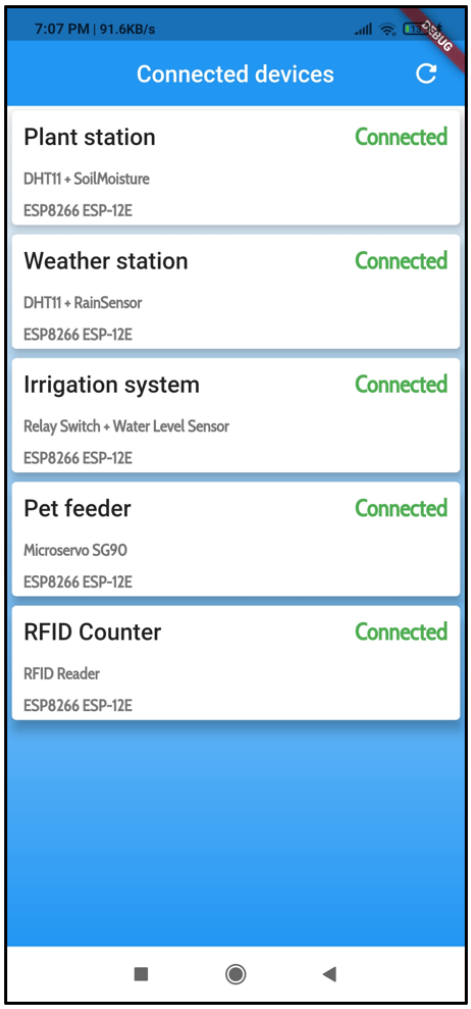
```
return Scaffold(
      resizeToAvoidBottomInset: false,
      appBar: AppBar(
        centerTitle: true,
        title: new Text('Connected␣devices'),
        elevation: 0.0,
        actions: [
          Padding(
            padding: EdgeInsets.only(right: 20.0),
            child: GestureDetector(
              onTap: () {
                _refreshAction();
              },
              child: Icon(
                Icons.refresh_rounded,
                size: 26.0,
              ),
```

```
          ),
        ),
      ],
    ),
    body: Container(
      decoration: BoxDecoration(
        gradient: LinearGradient(
          begin: Alignment.topCenter,
          end: Alignment.bottomCenter,
          colors: [
            Colors.white,
            Colors.blue,
          ],
        ),
      ),
      child: new Column(
        children: <Widget>[
          new Expanded(
            child: Container(
              child: futureBuilder,
            ),
          ),
        ],
      ),
    ),
  );
```

U "body" parametru prvo je definirana plavo-bijela pozadina aplikacije s efektom linearnog gradijenta. Nakon toga je započeto kreiranje prvih elemenata stabla widgeta od kojih je posljednji običan kontejner. Kontejner uvijek ima samo jedno dijete, a u ovom slučaju je to instanca prethodno opisanog "FutureBuilder" widgeta.

```
future: makecall.firebaseCalls(databaseReference)
```

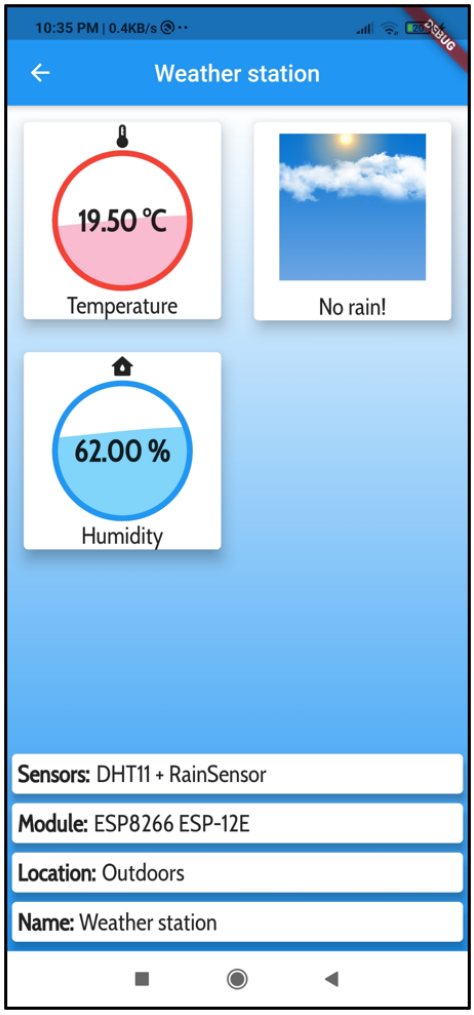
Najbitniji dio "FutureBuilder" widgeta je "future" parametar putem kojega se obavlja poziv na Firebase i dohvat svih potrebnih podataka. Dok se taj poziv ne izvrši i ne dohvate podaci, na zaslonu se ne može ništa ni prikazati. Zbog toga je dodana kružna animacija učitavanja, ali obično traje manje od sekunde. Podacima se upravlja instancom posebnog "AsyncSnapshot" widgeta koja u suštini predstavlja snimak baze. Nakon što je snimak dohvaćen moguće je obaviti iteraciju kroz sve module i prikazati ih kako je prikazano na slici 28, zajedno s pripadajućim podacima. Naglasak je na nazivu projekta, ispod kojeg su navedeni svi senzori i uređaji od kojih se projekt sastoji, kao i naziv mikrokontrolera i WiFi modula.



Slika 28: Aplikacija - Početni zaslon (Izvor: autorski rad)

Dizajn početnog zaslona dobiven je kombinacijom raznih widgeta, od kojih će biti spomenuti samo najbitniji. "Column" widget može imati više potomaka i svi će biti poredani vertikalno jedan ispod drugog, u stupac. Za razliku od "Columna", "Row" widget će djecu poredati horizontalno, u jedan redak. Za svaki modul se kreirao i poseban "Card" widget zbog čega su projekti odvojeni svaki u svoju karticu. Klikom na karticu se događa efekt širenja valova dobiven widgetom "InkWell". Ako je potrebno razdvojiti pojedine elemente, između njih se može dodati "SizedBox" widget kojem se odredi fiksna visina i širina. "Container" služi za lakše upravljanje i pozicioniranje svog widget djeteta jer ima mogućnost dodavanja margina, poravnanja, stilova i slično.

Početni zaslon sadržava i jedan gumb za osvježavanje u aplikacijskoj traci. Dodan je kako bi se na jednostavan način mogao osvježiti status svih uređaja. Naime, statusi "Connected" i "Not Connected" se ne dohvaćaju iz baze nego se određuju unutar same aplikacije. Iz tog razloga je dodan gumb kako bi u svakom trenutku bilo moguće provjeriti je li došlo do prekida veze između uređaja i baze podataka.



Slika 29: Aplikacija - Vremenska stanica (Izvor: autorski rad)

Klik na svaku karticu vodi do *dashboard* prikaza odabranog projekta. Programski kod zadužen za *dashboard* je u suštini isti za svaki projekt, dok konkretan prikaz ovisi o podacima iz baze. Na dnu ekrana prikazane su informacije o projektu, odnosno svi podaci koji se nalaze u "BasicInfo" segmentu u bazi podataka. Projekt vremenske stanice sa slike 29 se sastoji od dva tipa podataka: brojčanog i tekstualnog. Brojčani podaci se odnose na temperaturu i vlažnost zraka te su prikazani u lijevom stupcu. Tekstualni podatak vezan za intenzitet kiše prikazuje se u desnom stupcu.

Glavni dio *dashboarda* isprogramiran je u svrhu ostvarivanja potpune modularnosti. To znači da nije bitno koliko će brojčanih, numeričkih ili nekih drugih podataka biti spremljeno u bazu ili kako će ti podaci biti imenovani. Korisnik može imati sve tekstualne elemente napisane na nekom drugom jeziku, može u svom projektu imati desetke povezanih senzora ili nijedan. Svi će podaci biti prikazani u vlastitim karticama na *dashboardu* i raspoređeni u stupce ovisno o tipu. Stoga je jedina obaveza korisnika pratiti nekoliko jednostavnih pravila prilikom kreiranja skripte za mikrokontroler. Ako korisnik ispravno unese sve podatke u skriptu i uspješno ih pošalje na Firebase, oni će se instantno prikazati u aplikaciji.

```
return Column(
```

```
        children: [
            Expanded(
                child: Row(
                    children: [
                        if (_dbData.numericalReadings != null)
                            Expanded(
                                child: _futureBuilderNumerical(_dbData),
                        ),
                        if (_dbData.nonNumericalReadings != null)
                            Expanded(
                                child: _futureBuilderNonNumerical(_dbData),
                        ),
                        if (_dbData.interactive != null)
                            Expanded(
                                child: _futureBuilderInteractive(_dbData),
                        ),
                        if (_dbData.counter != null)
                            Expanded(
                                child: _futureBuilderRFIDCounter(_dbData),
                        ),
                    ],
                ),
            ),
            SizedBox(
                height: _dbData.basicInfo.length.toDouble() * 43,
                child: _futureBuilderBasicInfo(_dbData),
            ),
        ],
    );
```

Gornji isječak koda je zaslužan za pregledan prikaz svih elemenata na *dashboardu*. Pomoću kombinacije widgeta "Row" i "Expanded" ostvaren je prikaz elemenata jednakog tipa podataka po stupcima. Svaka "futureBuilder" metoda kao odgovor vraća listu pripadajućih elemenata u obliku stupca, dok je svaki element smješten u vlastitu karticu. Zbog toga se na slici 29 brojčani elementi prikazuju jedan ispod drugoga u lijevom stupcu, a tekstualni elementi u desnom stupcu. Svaki stupac će dobiti jednak postotak širine ekrana, ovisno o broju stupaca. Osnovni widget koji obuhvaća sve elemente pojedinog stupca je "ListView" widget. Zbog tog widgeta svaki stupac može imati varijabilan broj elemenata. Osim toga, omogućava klizno pomicanje svih elemenata u slučaju da svi elementi ne stanu na predviđeni dio zaslona, jednom riječju *skrolanje*.

Za sve brojčane elemente se ispisuju vrijednosti u sredini kartice te pripadajući nazivi u podnožju kartice. Vrijednosti su obogaćene korištenjem "LiquidCircularProgressIndicator" widgeta zbog vizualnog dojma. Ovaj animirani widget se sastoji od kruga s tekućinom, a količina ispune ovisi o samoj vrijednosti. Također, za brojčane elemente dodane su male ikone na vrhu kartice koje ovise o nazivu elementa.

Sve tekstualne vrijednosti iz baze će se prikazati u zasebnom stupcu, a u ovom radu će to biti samo podaci vezani uz senzor za kišu. Zbog toga su dodane i prigodne animacije iznad tekstualnih vrijednosti jednostavnom *switch-case* naredbom, u isječku koda ispod. Bu-

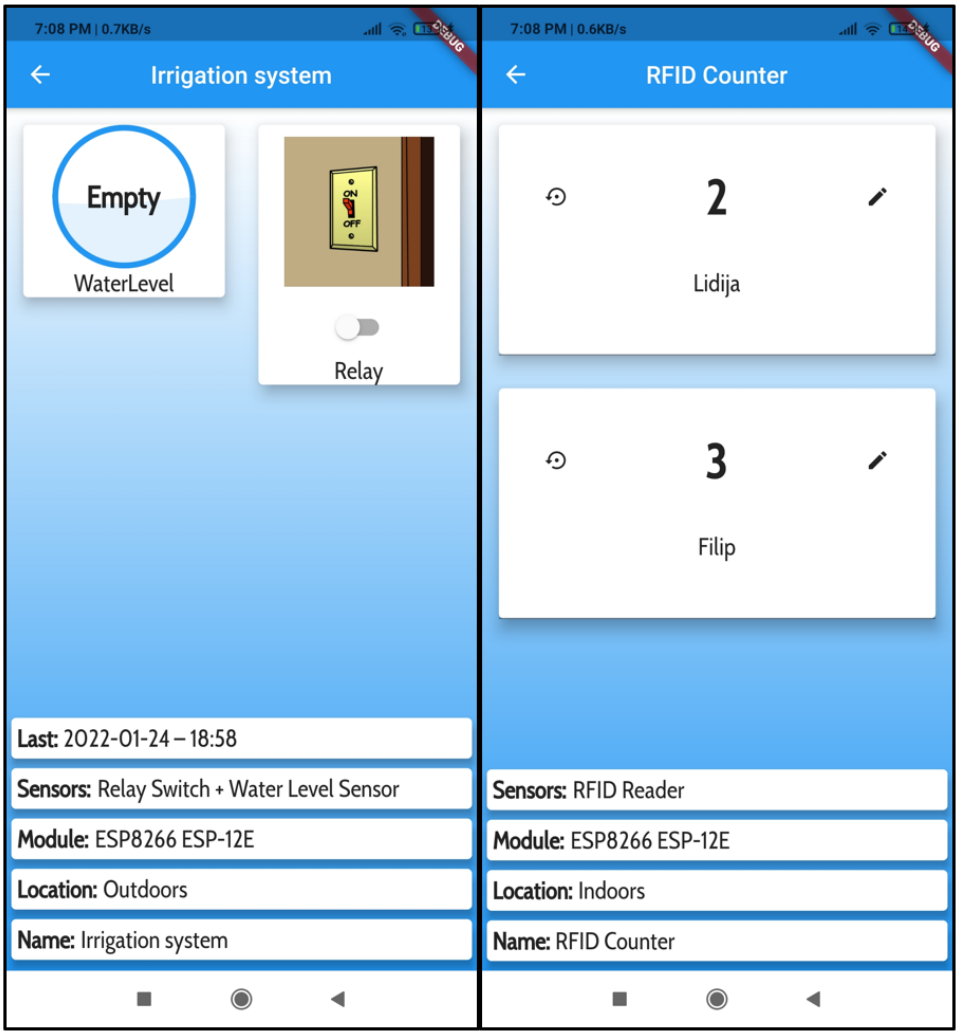
dući da su u Arduino skripti definirane četiri moguće vrijednosti senzora za kišu, dodano je pet animacija, po jedna za svaku vrijednost i jedna *defaultna*.

```
WeatherBg switchRain(ModuleDetails dbData, int index) {
    switch (dbData.nonNumericalReadings.values.elementAt(index)) {
        case 'No rain!':
            return WeatherBg(
                weatherType: WeatherType.sunny,
                width: 125,
                height: 125,
            );
        break;
        case 'Light rain!':
            return WeatherBg(
                weatherType: WeatherType.lightRainy,
                width: 125,
                height: 125,
            );
        break;
        case 'Medium rain!':
            return WeatherBg(
                weatherType: WeatherType.middleRainy,
                width: 125,
                height: 125,
            );
        break;
        case 'Heavy rain!':
            return WeatherBg(
                weatherType: WeatherType.heavyRainy,
                width: 125,
                height: 125,
            );
        break;
        default:
            return WeatherBg(
                weatherType: WeatherType.overcast,
                width: 125,
                height: 125,
            );
    }
}
```

Slika 30: Aplikacija - Stanica za biljke i hranilica za ljubimce (Izvor: autorski rad)

Dashboard stanice za biljke prikazan je na lijevoj strani slike 30 i sastoji se samo od brojčanih vrijednosti. Kartice pojedinih elemenata su automatski obuhvatile gotovo cijelu širinu ekrana jer ne postoje drugi tipovi podataka za prikaz. U ovom primjeru može se vidjeti i funkcionalnost *skrolanja* liste zato što je visina dobivenog stupca veća od visine slobodnog prostora na zaslonu.

Na desnoj strani se nalazi zaslon hranilice za ljubimce s jednim gumbom. U kodu aplikacije je definiran prikaz za dvije vrste interaktivnih elemenata: servo motor i prekidač. U ovom slučaju se radi o servu zbog čega se na zaslonu nalazi gumb. Dodana je i prigodna slika u svrhu poboljšanja vizualnog dojma i popunjavanja praznog prostora. U donjem dijelu zaslona, gdje su prikazane osnovne informacije, nalazi se i jedan dodatni podatak o vremenu posljednjeg aktiviranja hranilice.



Slika 31: Aplikacija - Relej sklopka i RFID čitač (Izvor: autorski rad)

Projekt s relej sklopkom, odnosno sustav za navodnjavanje, sastoji se od kombinacije brojčanih i interaktivnih elemenata. Interaktivni element je u ovom slučaju sklopka zbog čega se na zaslonu nalazi "Switch" widget. Iznad prekidača je i animacija paljenja i gašenja svjetla. Dodana je zato što je bitno korisniku dati do znanja kako prekidač ostaje u aktivnom stanju dok god ga se ne vrati na početno. Kao i kod drugih interaktivnih elemenata, u popisu osnovnih informacija se može pronaći i podatak o posljednjem paljenju prekidača. Taj podatak osigurava redovito navodnjavanje biljaka.

Element posebne vrste je RFID brojač čiji *dashboard* je prikazan na desnoj strani slike 31. Svaka kartica ovog tipa podataka sastoji se od gumba za resetiranje brojača s lijeve strane, prikaza vrijednosti brojača u sredini, gumba za uređivanje teksta s desne strane te teksta na dnu kartice. Svakim prislanjanjem RFID privjeska ili kartice će se korespondirajući brojač automatski povećati. Ako korisnik poželi koristiti RFID čitač za neku drugu namjenu, dovoljno je brojače resetirati na nulu i urediti tekst po želji.

6. Zaključak

U ovom radu opisani su najbitniji koncepti i karakteristike koji se vežu uz pojmove interneta i weba stvari. Navedene su najvažnije razlike između pojmova i razlozi zašto je web stvari bolja verzija interneta stvari. Bitno je napomenuti kako je broj uređaja povezanih na internet prestigao ukupnu ljudsku populaciju, što dovoljno govori o kakvoj rastućoj industriji je riječ. Internet stvari prati nove tehnološke inovacije i dosege te se sukladno tome neprestano razvija. Sasvim je prirodno je da će pojava tolikih razmjera, uz sve svoje benefite, imati i neke negativne posljedice.

Praktični dio rada prati razvoj jednostavnog pametnog doma koji se sastoji od pet neovisnih projekata. Unatoč tome što su neovisni, zahvaljujući Firebase platformi i web protokolima, centralizirani su u jednoj mobilnoj aplikaciji. Aplikacija je razvijena kombinacijom Dart jezika i Flutter okvira, a glavne značajke su modularnost i fleksibilnost. Ovaj rad može se zamisliti kao kostur ili temelj većeg sustava na koji je moguće dodati mnoge druge funkcionalnosti. Uz relativno malo programskog koda omogućen je prilagodljiv prikaz različitih podataka iz baze. Gomila funkcionalnosti koje Flutter pruža, brza krivulja učenja i interesantna tema učinile su izradu ovog rada vrlo poučnim i korisnim iskustvom.

Popis literature

[1] Wikipedia contributors, *Internet of things — Wikipedia, The Free Encyclopedia*, [Online; accessed 25-July-2021], 2021. adresa: https://en.wikipedia.org/w/index.php?title=Internet_of_things&oldid=1033647891.

[2] *Correcting the IoT History*, [Online; accessed 25-July-2021], 2017. adresa: http://www.chetansharma.com/correcting-the-iot-history/.

[3] ——, *Radio-frequency identification — Wikipedia, The Free Encyclopedia*, [Online; accessed 05-August-2021], 2021. adresa: https://en.wikipedia.org/wiki/Radio-frequency_identification.

[4] M. A. Bhabad i S. T. Bagade, „Internet of things: architecture, security issues and countermeasures," *International Journal of Computer Applications*, sv. 125, br. 14, 2015.

[5] Wikipedia contributors, *Microelectromechanical systems — Wikipedia, The Free Encyclopedia*, [Online; accessed 05-August-2021], 2021. adresa: https://en.wikipedia.org/wiki/Radio-frequency_identification.

[6] ——, *IPv6 — Wikipedia, The Free Encyclopedia*, [Online; accessed 05-August-2021], 2021. adresa: https://en.wikipedia.org/wiki/IPv6.

[7] ——, *Arduino — Wikipedia, The Free Encyclopedia*, [Online; accessed 06-August-2021], 2021. adresa: https://en.wikipedia.org/wiki/Arduino.

[8] ——, *Printed circuit board — Wikipedia, The Free Encyclopedia*, [Online; accessed 06-August-2021], 2021. adresa: https://en.wikipedia.org/wiki/Printed_circuit_board.

[9] R. Mitchell, *A Comparison of Popular Arduino Boards*, [Online; accessed 08-August-2021], 2018. adresa: https://maker.pro/arduino/tutorial/a-comparison-of-popular-arduino-boards.

[10] P. P., *A Beginner's Guide to the ESP8266*, [Online; accessed 08-August-2021], 2017. adresa: https://tttapa.github.io/ESP8266/Chap01%20-%20ESP8266.html.

[11] S. B. contributors, *NodeMCU, ESP12, ESP8266 - What is the difference?* [Online; accessed 19-January-2022], 2021. adresa: https://blog.spacehuhn.com/nodemcu-vs-esp8266/.

[12] J. Bonilla, *ESP8266 NodeMCU pinout for Arduino IDE*, [Online; accessed 14-January-2022], 2019. adresa: https://mechatronicsblog.com/esp8266-nodemcu-pinout-for-arduino-ide/.

[13] Wikipedia contributors, *Sensor — Wikipedia, The Free Encyclopedia*, [Online; accessed 06-August-2021], 2021. adresa: https://en.wikipedia.org/wiki/Sensor.

[14] *The Evolution of Consumer IoT*, [Online; accessed 28-July-2021], 2021. adresa: https://www.reply.com/en/topics/internet-of-things/the-evolution-of-the-consumer-internet-of-things.

[15] *IIoT- the Industrial Internet of Things (IIoT) explained*, [Online; accessed 29-July-2021]. adresa: https://www.i-scoop.eu/internet-of-things-guide/industrial-internet-things-iiot-saving-costs-innovation/industrial-internet-things-iiot/.

[16] H. Boyes, B. Hallaq, J. Cunningham i T. Watson, „The industrial internet of things (IIoT): An analysis framework," *Computers in industry*, sv. 101, str. 1–12, 2018.

[17] MIT, *Robotic gardening: MIT course creates robot-tending tomatoes*, [Online; accessed 04-August-2021], 2009. adresa: https://phys.org/news/2009-03-robotic-gardening-mit-robot-tending-tomatoes.html.

[18] P. Dong, Y. Han, X. Guo i F. Xie, „A Security and Safety Framework for Cyber Physical System," *2014 7th International Conference on Control and Automation*, 2014., str. 49–51. DOI: 10.1109/CA.2014.18.

[19] B. Buntz, *The Top 20 Industrial IoT Applications*, [Online; accessed 04-August-2021], 2017. adresa: https://www.iotworldtoday.com/2017/09/20/top-20-industrial-iot-applications/.

[20] A. Cuenca, *Spain's Santander hailed as global pioneering 'smart city'*, [Online; accessed 05-August-2021], 2016. adresa: https://www.thelocal.es/20160409/spains-santander-becomes-global-pioneering-smart-city/.

[21] K. Gyarmathy, *Comprehensive Guide to IoT Statistics You Need to Know in 2021*, [Online; accessed 11-August-2021], 2020. adresa: https://www.vxchnge.com/blog/iot-statistics.

[22] I. Ghosh, *4 key areas where AI and IoT are being combined*, [Online; accessed 11-August-2021], 2021. adresa: https://www.weforum.org/agenda/2021/03/ai-is-fusing-with-the-internet-of-things-to-create-new-technology-innovations/.

[23] S. Schenkluhn, *Market size and connected devices: Where's the future of IoT?* [Online; accessed 11-August-2021]. adresa: https://blog.bosch-si.com/internetofthings/market-size-and-connected-devices-wheres-the-future-of-iot/.

[24] K. Mišak, *Smrt transhumanizmu, sloboda narodu!* Zagreb: TELEdisk, 2020.

[25] D. Guinard i V. Trifa, *Building the Web of Things: With examples in Node.js and Raspberry Pi*. Manning Publications, 2016., ISBN: 9781638357094. adresa: https://books.google.hr/books?id=yzczEAAAQBAJ.

[26] W3C contributors, *Web of Things (WoT) Architecture*, [Online; accessed 15-August-2021], 2020. adresa: https://www.w3.org/TR/wot-architecture/#sec-architecture-overview.

[27] V. Barros, S. Junior, S. Bruschi, F. Monaco i J. Estrella, „An IoT Multi-Protocol Strategy for the Interoperability of Distinct Communication Protocols applied to Web of Things," listopad 2019., str. 81–88, ISBN: 978-1-4503-6763-9. DOI: 10.1145/3323503.3349546.

[28] Wikipedia contributors, *Node-RED — Wikipedia, The Free Encyclopedia*, [Online; accessed 16-August-2021], 2021. adresa: https://en.wikipedia.org/wiki/Node-RED.

[29] ——, *Firebase — Wikipedia, The Free Encyclopedia*, [Online; accessed 14-January-2022], 2021. adresa: https://en.wikipedia.org/wiki/Firebase.

Popis slika

Popis tablica