Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matic Repše

Digitalni nadzor akvaponskega sistema

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Luka Šajn

Ljubljana 2015

Fakulteta za računalništvo in informatiko podpira javno dostopnost znanstvenih, strokovnih in razvojnih rezultatov. Zato priporoča objavo dela pod katero od licenc, ki omogočajo prosto razširjanje diplomskega dela in/ali možnost nadaljne proste uporabe dela. Ena izmed možnosti je izdaja diplomskega dela pod katero od Creative Commons licenc http://creativecommons.si

Morebitno pripadajočo programsko kodo praviloma objavite pod, denimo, licenco *GNU General Public License, različica 3.* Podrobnosti licence so dostopne na spletni strani http://www.gnu.org/licenses/.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Študent naj v diplomskem delu predstavi strojne in programske rešitve pri razvoju digitalnega nadzora za potrebe akvaponskih sistemov. Predstavi naj izbor strojne opreme in njeno implementacijo. Poskrbi naj za upravljanje sistema na mestu in preko spletnega vmesnika. Opiše naj tudi razvoj spletne aplikacije za podporo upravljanju in nadziranju od ideje do rešitve ter navede tehnologije in programska orodja, ki jih je pri tem uporabil. V zaključnem delu naj poda možne poti nadaljnjega razvoja.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Matic Repše sem avtor diplomskega dela z naslovom:

Digitalni nadzor akvaponskega sistema

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Luka Šajna
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 28. avgusta 2015 Podpis avtorja:

Zahvaljujem se svojim najbližjim, ki so me tekom študija podpirali in spodbujali.
Zahvaljujem se tudi doc. dr. Luki Šajnu za pomoč pri nastajanju diplomskega dela.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
MCU	Microcontroller	mikrokrmilnik
PAR	Photosynthetically Active Radiation	fotosintetsko akvitno sevanje
DO	Dissolved Oxygen	raztopljeni kisik
JSON	JavaScript Object Notation	besedilna oblika zapisa za izmenjavo podatkov
API	Application Programming Interface	vmesnik za programiranje
AP	Access Point	dostopna točka
AJAX	Asynchronus JavaScript and XML	asinhroni JavaScript and XML
REST	Representational State Transfer	prenos stanj kot predstavitev
IDE	Integrated Development Environment	razvojno orodje
ACD	Analog to Digital converter	pretvornik iz analognega v digitalno
TTL	transistor-transistor logic	tranzistor-tranzistorska logika
SSID	Service Set Identifier	naziv brezžičnega omrežja
LCD	liquid crystal display	zaslon s tekočimi kristali
EEPROM	Electrically Erasable Programmable Read-Only Memory	električno zbrisljiv in programirljiv bralni pomnilnik
DRY	Don't repeat yourself	ne ponavljaj se

Povzetek

Cilj diplomskega dela je prikaz razvoja in implementacije cenovno ugodnejše strojne in aplikacijske rešitve za upravljanje in nadzor akvaponskih sistemov, pri katerih gre za hkratno pridelavo živalskih beljakovin (rib) in rastlin. V diplomskem delu smo tako predstavili akvaponiko kot metodo gojenja, njene sestavne elemente, ki zahtevajo našo pozornost, in problematiko, s katero se pri tem srečujemo, čemur smo dodali opis in kratek pregled izbranega pristopa k rešitvi. V nadaljevanju smo predstavili na eni strani izbiro ter na drugi strani razvoj strojne opreme in s primeri ponazorili programsko implementacijo slednje. Nadaljevali smo z opisom izbranih tehnologij, namenjenih razvoju spletne aplikacije, ki služi kot podpora strojnemu delu rešitve ter predstavili njene glavne funkcionalnosti in njihov razvoj, kar smo podprli s primerom uporabe. V zadnjem delu smo navedli možnosti nadaljnje razširitve strojnega in aplikacijskega dela.

Ključne besede: akvaponika, strojna oprema, spletna aplikacija, mikrokrmilnik, Rails.

Abstract

The goal of the thesis is to show the development and implementation process of hardware and software solution for controlling and managing Aquaponics systems, which are responsible for simultaneous production of animal protein (fish) and plants. In the thesis, we described what Aquaponics consists of, where is our attention mostly needed and the problems we have to deal with. We also added a short explanation of the possible corresponding approach to a solution. Hereafter, we presented our choice for already existing hardware equipment, as well as the development of our own. We've supported everything with examples of code. Below we explained what technologies we've used for development of web application and described its main functionalities and implementation of those, what we've supported with example of usage. In the last part we specified further possible extensions for both hardware and application part of the solution.

Keywords: Aquaponics, hardware, web aplication, microcontroller, Rails.

Poglavje 1

Uvod

Zadnja leta se zaradi širjenja okoljske ozaveščenosti, krčenja naravnih virov oziroma njihovega pomanjkanja v vse večjem obsegu uvajajo in razvijajo trajnostni načini pridelave hrane. To pomeni, da se osredotočamo na večji odstotek pridelka in predvsem njegov manjši odpad, kot tudi manjši odpad vode. Ena tovrstnih metod je akvaponika, ki je bila pozabljena, a nedavno obujena in posodobljena. Gre za hkratno pridelavo živalskih beljakovin - rib ter rastlin - predvsem v obliki zelenjave in začimbnic. Glavna prednost akvaponske metode gojenja je zaprt krožni vodni sistem, kar pomeni, da je izredno učinkovita in še posebej primerna v krajih s sušnim podnebjem, a ima tudi pomanjkljivosti.

Zaprtost sistema, ki je ne glede na velikost, v primerjavi z obdajajočim okoljem relativno majhen, v tem primeru lahko pomeni izrazito nihanje stanj posameznih parametrov v krajšem časovnem obdobju (nekaj ur), kar je lahko usodno za celoten sistem. Da bi se takim primerom v čim večjem obsegu izognili, potrebujemo učinkovit nadzor nad parametri in dostop do ves čas osveženih podatkov, ki predstavljajo trenutno stanje sistema. To lahko hkrati predstavlja tudi večjo finančno investicijo, če se odločimo za nakup končnega izdelka, ki je namenjen specifični uporabi. Dandanes se ob poplavi spletnih strani, ki po izjemno nizki ceni ponujajo skoraj vso opremo, primerno za elektronske meritve, temu lahko izognemo. Naš cilj je bil razviti cenovno

ugodno celovito rešitev, ki bi se navezovala tako na strojni, kar vključuje izbor in razvoj opreme za nadzor, kot tudi aplikacijski del, ki skrbi za beleženje in prikaz zajetih vrednosti, pa tudi za nadzor nad parametri in obveščanje ob kritično mejnih primerih.

V diplomskem delu smo sprva bolj natančno opredelili akvaponiko in pomen parametrov, ki nastopajo v njej. Nato smo predstavili izbor in implementacijo strojne opreme za potrebe nadzora akvaponskega sistema. Sledi kratek opis strežniških tehnologij za lažje razumevanje končnega dela, kjer smo opisali razvoj in najpomembnejše funkcionalnosti spletne aplikacije. Za konec smo predstavili primer uporabe in možne nadaljnje razširitve, ki bi dodatno pripomogle k učinkovitemu upravljanju akvaponskega sistema.

Poglavje 2

Analiza področja

2.1 Akvaponika

Kmetijsko pridelovalno panogo, ki jo danes poznamo pod pojmom akvaponika, so v nekoliko drugačni obliki poznali že tisočletja pred nami. Prvi zapisi na temo akvaponike segajo v čas obstoja Aztekov, ki so za potrebe pridelave hrane postavili sistem plavajočih vrtov, t.i. "chimpas" [11]. Tekom življenja je bila ta metoda gojenja za nekaj stoletij pozabljena, v sodobnem času pa se je ponovno odkrita najprej razvila v Avstraliji zaradi toplejšega in bolj suhega podnebja ter slabše pridelovalne kakovosti tal. Iz hobija vrtičkarjev se je razvila v hitro rastočo industrijo. Evropa je šele pričela z uvajanjem te "nove" tehnologije, medtem ko največje tovrstne sisteme najdemo na Japonskem in v Združenih Arabskih Emiratih [1].

Akvaponika(ang. aquaponics)pomeni povezovanje ribogojstva (ang. aquaculture) s hidroponiko(ang. hydroponics) [6].

Hidroponika pomeni vzgajanje rastlin brez prisotnosti prsti (ang. soilless culture), le to nadomešča rastni medij, ki rastlini nudi oporo ter zadostno vlago. V to metodo gojenja je dodatno integriran namakalni (pretočni) sistem, ki rastlinam prinaša vsa umetno dodana potrebna hranila za njihovo rast [11]. Nadaljnje poznamo NFT (ang. nutrient film technique) različico pretočnega sistema, kjer zelo plitev tok vode teče mimo korenin, ki se naha-

jajo v rastnem mediju. To je tudi sistem, ki se ga uporablja v akvaponiki. Hidroponika je nepogrešljiva na območjih, kjer zemlje ni mogoče obdelovati, njena največja težava pa so umetno dodana gnojila.

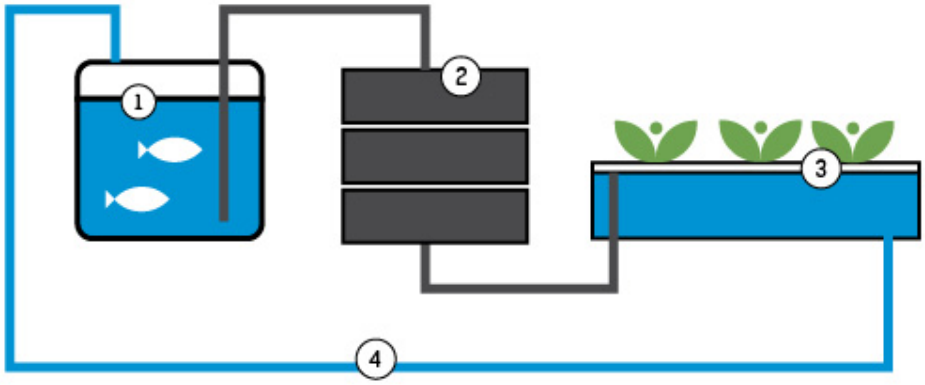
Akvakultura označuje gojenje vodnih organizmov v umetnem okolju pod nadzorovanimi pogoji in postaja vse bolj pomemben vir proizvodnje proteinov, saj se je že leta 2009 odstotek gojenih rib izenačil z ulovom [3]. Tradicionalno gojenje vodnih organizmov potrebuje ogromno vode, saj moramo zaradi izločkov vodo redno menjavati. Obstaja pa tudi metoda RAS (ang. recirculating aquculture system), ki je zelo učinkovita pri varčevanju z vodo, saj se ta ponovno uporabi po procesih čiščenja in filtriranja. Ostanek predstavlja odpadna voda, ki v akvaponiki izgubi pomen odpadna.

Akvaponika povezuje akvakulturo, ki uporablja kroženje vode (RAS), skupaj s hidroponiko (NFT) v en proizvodni sistem. V akvaponskem krogu (Slika 2.1) voda iz bazena z ribami(1) kroži najprej skozi filter(2), nato preko gojišča rastlin(3) in nazaj v bazen z ribami(4). Filter pravzaprav tvorita dve različni komponenti. Prva je mehanski filter, ki iz vode odstrani ribje izločke v trdi obliki. Drugi filter je t. i. biofilter, ki predela raztopljene odpadke. V biofiltru se nahajajo bakterije, ki pretvarjajo amonijak, ki je toksičen za ribe, v nitrate, ki so hranilo za rastline. Proces se imenuje nitrifikacija. Z nitrati obogatena voda potuje mimo korenin rastlin, kjer jo rastline osiromašijo, nakar se prečiščena vrne v rezervoarje z ribami [6]. Celoten proces omogoča, da ribe, rastline in bakterije uspevajo simbiotično, skupaj ustvarijo rastno okolje drug za drugega in vzpostavijo ravnotežje.

S tem ko ponovno uporabimo vodo in posledično rastline pridobijo hranila, odstranimo faktorje netrajnosti, ki jih posamezno gledano najdemo v akvakulturi in hidroponiki. Produkt pri akvaponiki je v boljših sistemih tako rekoč enak kot skupni pri panogah, iz katerih izhaja. Akvaponika je še posebej ekonomično upravičeno in lažje izvedljiva v krajih, kjer primanjkuje vode.

Pozitivni vidiki akvaponike:

- trajnostna rešitev za pridelavo hrane;



Slika 2.1: Akvaponski krog (avtor: Matej Leskovec)

- dva različna pridelka (zelenjava in ribe);
- izjemno učinkovita pri ohranjanju vode;
- ne potrebuje prsti;
- ne uporablja pesticidov in gnojil.

Negativni vidiki akvaponike:

- drag začetek;
- potrebno znanje o ribogojstvu, bakterijah in pridelavi zelenjave;
- popolne razmere za ribe in rastline je včasih težko ustvariti;
- ni primerna na območjih, kjer razpon temperature ni primeren za rastline ali ribe;
- manj možnosti za upravljanje v primerjavi s samostojnimi sistemi (akvakultura in hidroponika);
- napake lahko povzročijo propad sistema;

- potrebno je dnevno vodenje/upravljanje;
- potreba po električni energiji.

2.1.1 Pomembni parametri v akvaponskem sistemu

pH

pH vode ima neposredni učinek na biološko aktivnost nitrificirnih bakterij in s tem na njihovo zmožnost pretvarjanja amoniaka v nitrit. Bakterijam ustreza nekoliko višji pH kot rastlinam, zato v primeru, ko ga prilagodimo slednjim, zmanjšano dejavnost bakterij nadomestimo z večjim biofiltrom.

S tem ko merimo pH (ang. potential of hydrogen) v resnici merimo koncentracijo vodikovih ionov $[H^+]$ v raztopini. Več kot je vodikovih ionov, bolj je neka raztopina kisla in obratno, manj kot jih je, bolj je bazična. pH je definiran kot negativen logaritem koncentracije vodikovih ionov.

$$pH = -log[H^+] \tag{2.1}$$

, kjer je $[H^+]$ koncentracija vodikovih ionov v mol/L. pri čemer se moramo zavedati, da je pH skala:

- negativna; pH 7 ima manj vodikov ionov kot pH 6;
- logaritmična; pH 7 ima 10-krat manj vodikov ionov kot pH 6, 100-krat manj kot pH 5.

Logaritmična skala nas lahko hitro zavede, tako npr. za idealen pH 7 v sistemu ob meritvi pH 9 pomeni, da je problem lahko 100-krat večji in ne 2-krat. Prilagajanje pH-ja v sistemu je relativno enostavno. Kadar imamo prenizek pH, dodajamo bazične raztopine in nasprotno, kadar imamo previsok pH, dodajamo kisle raztopine.

Najbolj pogost način merjenja poteka s pomočjo pH sonde, ki omogoča stalno, kontinuirano in natančno merjenje, poleg tega omogoča tudi elektronsko beleženje rezultatov. V nadaljevanju je opisano, kako taka sonda deluje,

v prihajajočem poglavju pa je opisan tudi postopek izdelave lastnega vezja za nadzor pH.

PH sonda je naprava, ki deluje podobno kot baterija z enojno celico z zelo veliko upornostjo. Inducirana napetost na elektrodi je sorazmerna z vsebnostjo vodikovih ionov, ki se nahajajo v okolici merilne elektrode, odvisna pa je od razlike v potencialih med merilno in referenčno elektrodo [14].

$$E = E_m - E_r \tag{2.2}$$

Delovanje elektrod opisuje Nernstova enačba, ki opisuje razmerje med napetostjo in sestavo galvanskega člena.

$$E = E_0 - \frac{2.3RT}{nf} log a_i \tag{2.3}$$

, kjer
$E = potencial \quad med \quad elektrodama$
$E_0 = standardni \quad potencial \quad iona$
$2.3RT/nf = Nernstov \quad faktor$
$a_i = aktivnost \quad iona$

Nernstov faktor nam pove, za koliko se spremeni potencial ob desetkratni spremembi koncentracije ionov (1 pH enota). Kjer je n = 1 in T = 25 °C za idealno sondo to znaša 59.16 mV. Nernstov faktor se spreminja s temperaturo – ko slednja narašča, z njo raste tudi napetost.

$$E = E_0 + (1.98 \times 10^{-4}) \times T_K pH \tag{2.4}$$

Glavne značilnosti:

- pri pH 7 je izhod sonde 0 V, pri pH $>$ 7 je napetost pozitivna, pri pH $<$ 7 je napetost negativna;
- če sonda ustvari 59 mV/pH enoto, je efektivni razpon +/-7*59 mV ali +/- .414 V;

- v odvisnosti od temperature se generirana napetost spreminja od 54 mV/pH enoto pri 0 °C do 74 mV/pH enoto pri 100 °C.

Temperatura vode

Temperatura vode je pomemben faktor za bakterije in ima vsesplošen vpliv (toksičnost, raztopljen kisik, vsrkavanje hranil) v akvaponskem sistemu. Popolna temperatura vode je približno 20 °C, če pade pod 17 °C, začne produktivnost strmo padati, pri 10 °C je manj kot 50%. Zimske nizke temperature imajo zato velik vpliv na upravljanje sistema pozimi. Tudi visoke temperature zahtevajo pozornost, slednje namreč škodujejo predvsem ribam.

Raztopljen kisik

Kisik je ključnega pomena za vse tri organizme, prisotne v akvaponiki. Je tisti parameter vode, ki v najkrajšem času najbolj drastično vpliva na spremembe. Ribe lahko zaradi prenizke vrednosti umrejo v nekaj urah, nitrificirne bakterije pa močno spustijo nivo produktivnosti. Ker so sonde za merjenje raztopljenega kisika razmeroma drage, je za manjše sisteme včasih dovolj, da opazujemo vzorec obnašanja rib in splošno stanje rastlin, hkrati moramo poskrbeti, da zračne in vodne črpalke delujejo ves čas in tako skrbijo za vnos kisika in kroženje vode, ki sta ključna mehanska procesa v akvaponiki. V naravnem okolju, kjer ni intenzivnega vzgajanja rastlin, kisik pride v vodo preko vodnega površja v dovolj velikih količinah, da ribe lahko preživijo.

Temperatura vode in vsebnost kisika sta neposredno povezana. S tem ko temperatura vode narašča, sposobnost/kapaciteta vsebnosti kisika pada. Ali drugače, v hladni vodi je več kisika kot v toplejši. Iz tega neposredno sledi, da morajo v toplih mesecih in na toplih območjih črpalke delovati intenzivnejše ali pa moramo povečati njihovo število.

PAR, alge

Alge v sistemu vplivajo na kakovost vode, in sicer na pH, raztopljen kisik in dušik. Alge spadajo v razred fotosintetičnih organizmov podobno kot rastline

in uspevajo v vsaki vodi, ki je bogata s hranili in izpostavljena sončni svetlobi. V akvaponskem sistemu želimo rast alg zatreti zaradi več razlogov. Iz vode vsrkavajo hranila in tako tekmujejo s ciljnimi rastlinami. V sistemu delujejo kot porabnik in proizvajalec kisika, tekom dneva ga s fotosintezo proizvajajo, ponoči ga z dihanjem porabljajo. Koncentracija DO lahko ponoči zaradi aktivnosti alg pade pod kritično mejo, kar je neposredno povezano z ogljikovim dioksidom, slednje pa neposredno vpliva na pH v sistemu. Podnevi se odstranjuje ogljikova kislina, pH narašča in nasprotno ponoči. Vplivajo tudi na mašenje odtokov in tako otežujejo kroženje vode. Razvoj alg lahko zatiramo z zasenčitvijo vodnih površin oz. tako, da nadziramo vrednosti fotosintetične aktivne radiacije (ang. photosynthetically active radiation) in s tem koliko svetlobe, ki vpliva na rast, pride do oz. pod vodno površino.

Vir vode

V povprečju akvaponski sistem na dan izgubi 1-3% vode, odvisno od okolja in rastlin, ki jih gojimo. Nekaj vode se izgubi preko rastlin, ostalo zaradi neposrednega izhlapevanja in gibanja (pršenje) rib. Vodo je zato potrebno periodično dovajati v sistem.

2.1.2 Motivi za razvoj lastnega digitalnega nadzora

Nekatere od negativnih vidikov akvaponike lahko omilimo z lastnim pristopom. Tako lahko npr. določene dele strojne opreme s pomočjo znanja, ki nam je na voljo v literaturi ali na spletu, sestavimo oz. razvijemo sami in s tem drastično znižamo stroške. Enako velja za programske rešitve. Profesionalni izdelki, ki so na trgu, pogosto ne ponujajo nič več kot lahko naredimo sami, poleg tega le malokrat nudijo možnost povezave v omrežje ali pa je za vzpostavitev potreben nakup dodatnih dragih modulov. S povezavo z omrežjem pridobimo predvsem na varnosti sistema v smislu dovolj hitrega odzivanja na javljene napake, ki se lahko prožijo preko spletne pošte ali pa SMS-a. Še ena pozitivna lastnost omreženja je ta, da lahko vrednosti parametrov spremljamo oddaljeno, zato nismo primorani vsakodnevno ročno na

mestu upravljati s sistemom. Popolne razmere za ribe in rastline je včasih težko ustvariti, a z nenehnim nadzorom lahko testiramo proces in se jim približamo. Z vzpostavitvijo digitalnega nadzora lahko posredno vplivamo tudi na porabo električne energije. Primer: s prezračevanjem spreminjamo vlažnost v prostoru, vlažnost vpliva na količino energije, ki je potrebna za segrevanje zraka, če imamo torej podatek o zunanji in notranji vlažnosti, se lahko sistem pred segrevanjem odloči, ali je smiselno prezračevanje in s tem zmanjša porabo energije.

2.2 Digitalni nadzor

Digitalni nadzor oz. avtomatizacijo in posledično mikrokrmilnike dandanes najdemo na veliko mestih, nahajajo se npr. v hladilnikih, mikrovalovnih pečicah, telefonih, sesalcih, tako rekoč vsepovsod okoli nas, a se tega le redko zavedamo. Kljub temu, da so nezaznavni, pa so očitno zelo pomembni. Nastanek mikrokrmilnikov je bil pogojen sprva z izumom elektronk (ang. valve tube), ki so jih nasledili manjši tranzistorji. Z manjšanjem tranzistorjev in kopičenjem več tranzistorjev v posamezni čip so nastala prva logična vrata ali vezja. Manjši kot so postajali, več komponent je bilo mogoče namestiti v posamezni čip, tako so nastali prvi mikrokrmilniki.

Mikrokrmilnik je integrirano vezje, ki je sprogramirano, da izvaja določeno nalogo in je po svoje zelo majhen računalnik. Integrirano vezje (ang. integrated circuit) ali čip je naprava, izdelana iz polprevodniških elementov. Pomembno je, da med seboj razlikujemo in ne mešamo pojma mikrokrmilnik s pojmom mikroprocesor, razlike med njima so predstavljene v Tabeli 2.1.

Mikrokrmilniki se torej uporabljajo za specifične naloge. Ne potrebujejo močnega procesorja, saj za izpolnitev nalog po navadi zadošča zgolj nekaj MHz in malo prostora. Da MCU postane koristen, ga je predhodno potrebno programirati. Danes se za to večinoma uporablja razvojna orodja (IDE),

	Mikroprocesor	Mikrokontroler
Vmestitev	Računanje (tablični računalniki, prenosniki)	Aparati, specializirane naprave
Hitrost	Hiter (3 GHz)	Relativno počasen (16 MHz)
Zunanji deli	Veliko (samo računa, ne vsebuje spomina)	Malo (samostojna naprava)
Cena	Drag (več kot 100$)	Poceni(manj kot 10$)
Poraba energije	Visoka	Nizka
Proizvajalci	Intel, AMD, ARM	Atmel(AVR), Microchip, Texas Instruments

Tabela 2.1: Primerjava med mikroprocesorjem in mikrokrmilnikom

programiranje pa poteka v zbirnem jeziku (ang. assembler), C ali C++. Razvojno orodje preveri pravilnost programa in ga prevede (ang. compile) v binarno kodo, razumljivo MCUju. Za prenos kode na MCU se uporablja t.i. programator (ang. programmer). Pogost način programiranja čipov, ki se uporablja, je ICSP(ang. in-circuit serial programmer), pomeni pa, da čip lahko programiramo tudi naknadno po tem, ko je vgrajen v nek končni sistem in ne le v času proizvodnje [13].

Pomembna lastnost MCUjev so vhodno/izhodne operacije, ki jih podpirajo. Zelo pomemben element je ADC (ang. Analog to Digital converter), vsebuje ga skoraj vsak novejši MCU. Glavna lastnost ADC je pretvorba analognih signalov (npr. napetosti) v digitalne vrednosti, npr. digitalna predstavitev napetosti. To je pomembno, ker lahko digitalne vrednosti analiziramo v programu, zapišemo v spomin in z njimi operiramo.

Poglavje 3

Uporabljene tehnologije, orodja in strojna oprema

3.1 Strojna oprema

3.1.1 Testni sistem

Testni sistem (Slika 3.1), ki je del predstavljene rešitve, se v času pisanja nahaja na Biotehniški fakulteti v plastenjaku. Sestavljata ga 2 ločena bazena z ribami, gojišče z glinoporjem in gojišče na vodi. V obeh bazenih z ribami se ločeno nadzira temperaturo, pH vrednosti, PAR vrednosti in dovod uravnalnih tekočin. Spremljata se tudi temperatura in vlaga zraka, nad enim od bazenov pa se nahaja avtomatski hranilnik, ki ga krmili mikrokrmilnik.

3.1.2 Mikrokrmilnik

Centralni element v nadzorni enoti predstavlja Atmelov 8-bitni AVR ATmega2560 mikrokrmilnik, ki temelji na RISC (ang. reduced instruction set computing, poenostavljen nabor ukazov) arhitekturi. Slednja predvidoma nudi boljšo zmogljivost na MCUjih, ki podpirajo tovrstno arhitekturo, v primerjavi s predhodno CISC (ang. complex instruction set computing) arhitekturo. MCU, ki je uporabljen v predstavljeni rešitvi, se nahaja na Arduino

Slika 3.1: Testni sistem

Mega 2560 (Slika 3.2) razvojni ploščici (ang. development board); slednjo smo izbrali, ker vsebuje vse potrebne elemente za delovanje in omogoča hitrejši razvoj v razvojni fazi v primerjavi s primerom, kjer bi razvili svoje vezje, ki bi nudilo le to, kar potrebujemo. Dotična ploščica nam dodatno ustreza, ker vsebuje veliko število vhodno/izhodnih nožic, večje število prekinitev in več strojnih serijskih povezav, kjer velja, da lahko eno od teh preusmeri preko čipa ATmega16U2, ki omogoča povezavo z računalnikom in programiranje MCU brez vmesnega programatorja [2]. Slednje nam hkrati olajša reševanje napak, saj lahko preko serijske povezave na računalniku spremljamo dogajanje na MCU in tako uspešno odstranjujemo hrošče. Tehnična specifikacija za Arduino Mega 2560 je prikazana v Tabeli 3.1. Na tem mestu omenimo še, da se pri programiranju Arduina skoraj vedno opremo na funkciji setup() in loop(). Namen prve je, da se izvede ob priklopu naprave in posledično vse, kar je definirano znotraj nje, ko se konča, se kliče druga, ta se nenehno vrti v zanki, kar nakazuje že njeno ime.

Mikrokrmilnik	ATmega2560
Obratovalna napetost	5V
Napajalna napetost	7-12V
Digitalni V/I pini	54
Analogni vhodni pini	16
Maksimalen tok na pin	20mA
Flash spomin	256 KB
SRAM	8 KB
EEPROM	4 KB
Hitrost procesorja	16 MHz

Tabela 3.1: Tehnične značilnosti Arduino Mega 2560

Slika 3.2: Arduino Mega 2560 razvojna ploščica

3.1.3 LCD zaslon

LCD zaslon predstavlja edino interakcijo med uporabnikom in strojno opremo v realnem času na mestu nadzora. Da je to mogoče, smo izbrali 3.2 inčni TFT LCD zaslon z vgrajenim SSD1289 krmilnikom in nameščenim modulom za dotik. Tovrstna izbira nam je močno razširila možnosti za implementacijo nadaljnjih funkcionalnosti, kot so proženje črpalk in hranilnika z nastavljivo časovno omejitvijo, kalibracija parametrov, napredna kalibracija pH, izbira povezljivosti v omrežje... S tem smo se znebili morebitnih dodatnih fizičnih gumbov in vnovičnem programiranju za vsako željeno spremembo. Slaba stran izbranega zaslona pa je, da za povezavo z MCUjem zasede kar 30 nožic, med njimi večinoma vhodno/izhodne in tiste, ki skrbijo za napajanje. Programiranje prikaza poteka preko osnovnih funkcij, ki skrbijo za izris krogov, pikslov, pravokotnikov, teksta... Z vsemi naštetimi in preostalimi funkcijami smo si pomagali, da smo izrisali menije in gumbe. Ker je risanje gumbov pogosto in ohranjamo enak slog, smo implementirali svojo funkcijo drawButton() (Primer 3.1), ki za parametre sprejme pozicijo in velikost gumba, tekst in položaj besedila.

```
void draw_button(int x1, int y1, int x2, int y2, String text, int x_t,
    int y_t)
{
  LCD.setFont(SmallFont);
  LCD.setColor(11, 72, 107);
  LCD.fillRoundRect(x1, y1, x2, y2);
  LCD.setColor(255, 255, 255);
  LCD.setBackColor(11, 72, 107);
  LCD.drawRoundRect(x1, y1, x2, y2);
  LCD.print(text, x_t, y_t);
}
```

Primer 3.1: Definicija funkcije za risanje gumbov

Prav tako smo implementirali funkcijo screen_info(), slednja sprejme parametre za besedilo, položaj besedila, vrsto pisave in ukaz za brisanje zaslona. Namenjena je torej pisanju besedila na zaslon.

```
void screen_info(String text, int x_position, int y_position, boolean
     clear, uint8_t *font)
{
  if (clear)
  {
    LCD.fillScr(59, 134, 134);
  }
  LCD.setFont(font);
  LCD.setColor(255, 255, 255);
  LCD.setBackColor(59, 134, 134);
  LCD.print(text, x_position, y_position);
}
```

Primer 3.2: Definicija funkcije za izpis besedila na zaslonu

Kako obvladujemo dogodek, ko se uporabnik dotakne zaslona, si poglejmo na primeru, ko uporabnik klikne gumb za WiFi povezavo. Da bi razumeli položaje, omenimo še, da smo orientacijo ekrana definirali kot pokončno, enako velja za površino, občutljivo na dotik.

```
LCD.InitLCD(PORTRAIT);
TOUCH.InitTouch(PORTRAIT);
```

Primer 3.3: Določitev orientacije

Ker ima ekran mrežo 320 x 240 pikslov, to pomeni, da je v pokončnem načinu os x (horizontala) dolga 240 pikslov, os y (vertikala) pa 320 pikslov.

```
draw_button(70, 165, 170, 200, "WiFi", 105, 177);
```

Primer 3.4: Uporaba funkcije drawButton()

Zgornji levi del gumba se nahaja na 70. pikslu od leve proti desni in 165 pikslu od zgoraj navzdol, širok je 100 in visok 35 pikslov. Da bi zaznali pritisk dotičnega gumba, se v loop() kliče funkcijo touch() (Slika 3.3), ki smo jo implementirali tako, da se na prvem nivoju vpraša, če se je zgodil dotik, na drugem, kateri od menijev se trenutno nahaja na zaslonu, na tretjem kje v horizontalnem območju se je zgodil dotik in nazadnje kje v vertikalnem območju. Če imamo izpolnjen pogoj na vsakem nivoju, ukrepamo naprej.

```
if sense touch then
    read x and y coordinates;
    match result with currently shown menu;
    if find predefined matching vertical position then
        if find predefined matching horizontal position then
            perform action;
        end
    end
end
```

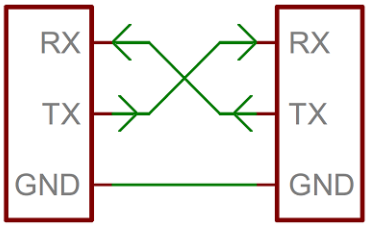
Slika 3.3: Ugotavljanje koordinat dotika

Zaslon je tudi edini vir informacij na mestu nadzora, zato na njem prikazujemo vse trenutno merjene vrednosti, ki se periodično osvežujejo (Slika 3.4).

3.1.4 Dostop do omrežja

Dostop do omrežja smo omogočili preko ethernet in brezžične povezave. Za ethernet skrbi Ethernet razširitveni modul, ki je združljiv z Arduino razvojno ploščico, na katero ga fizično namestimo. Za komuniciranje z modulom smo uporabili Ethernet knjižnico, ki podpira vse potrebne osnovne ukaze oz. protokole za dostop in komunikacijo preko omrežja. Poleg žične povezave smo omogočili tudi brezžično ali WiFi povezavo, ki se vrši preko modula ESP8266. Slednjega smo med ostalimi možnostmi izbrali na podlagi dejstev, da je poceni, majhen, programsko nadgradljiv in poseduje dobre karakteristične lastnosti povezovanja v omrežje. Deluje neodvisno od mikrokrmilnika in za delovanje uporablja svoje programje, ki je shranjeno v pomnilniku. MCU in modul medsebojno komunicirata preko asinhrone serijske povezave (TTL), tj. preko nožice TX, ki je namenjena oddajanju (ang. transmit) in RX, ki je namenjena sprejemanju (ang. receive). Logično sledi, da je nožica TX z Arduina povezana na nožico RX modula, enako velja za preostale (Slika 3.5).

Slika 3.4: Prikaz vrednosti na zaslonu



Slika 3.5: Asinhrona serijska povezava

Asinhrona serijska povezava je metoda za prenos podatkov, ki za svoje delovanje potrebuje kar najmanjše število fizičnih povezav in posledično vhodno/izhodnih priklopov. Za razliko od sinhrone ne uporablja pomoči zunanjega urinega signala, potrebuje pa dodatne mehanizme v implementaciji, da zanesljivo pošilja in prejema informacije.

Za osnovno postavitev ESP8266 je potrebno naslednje sosledje ukazov, ki jih pošiljamo:

1. AT+RST (ponastavimo modul, odgovor: OK);
2. AT+CWMODE=1 (STA način delovanja, povezovanje na dostopno točko, odgovor: OK);
3. AT+CWJAP=ŠSID”,”pass”(podamo SSID omrežja in geslo, odgovor: OK);
4. AT+CIFSR (odgovor: pridobljen IP naslov).

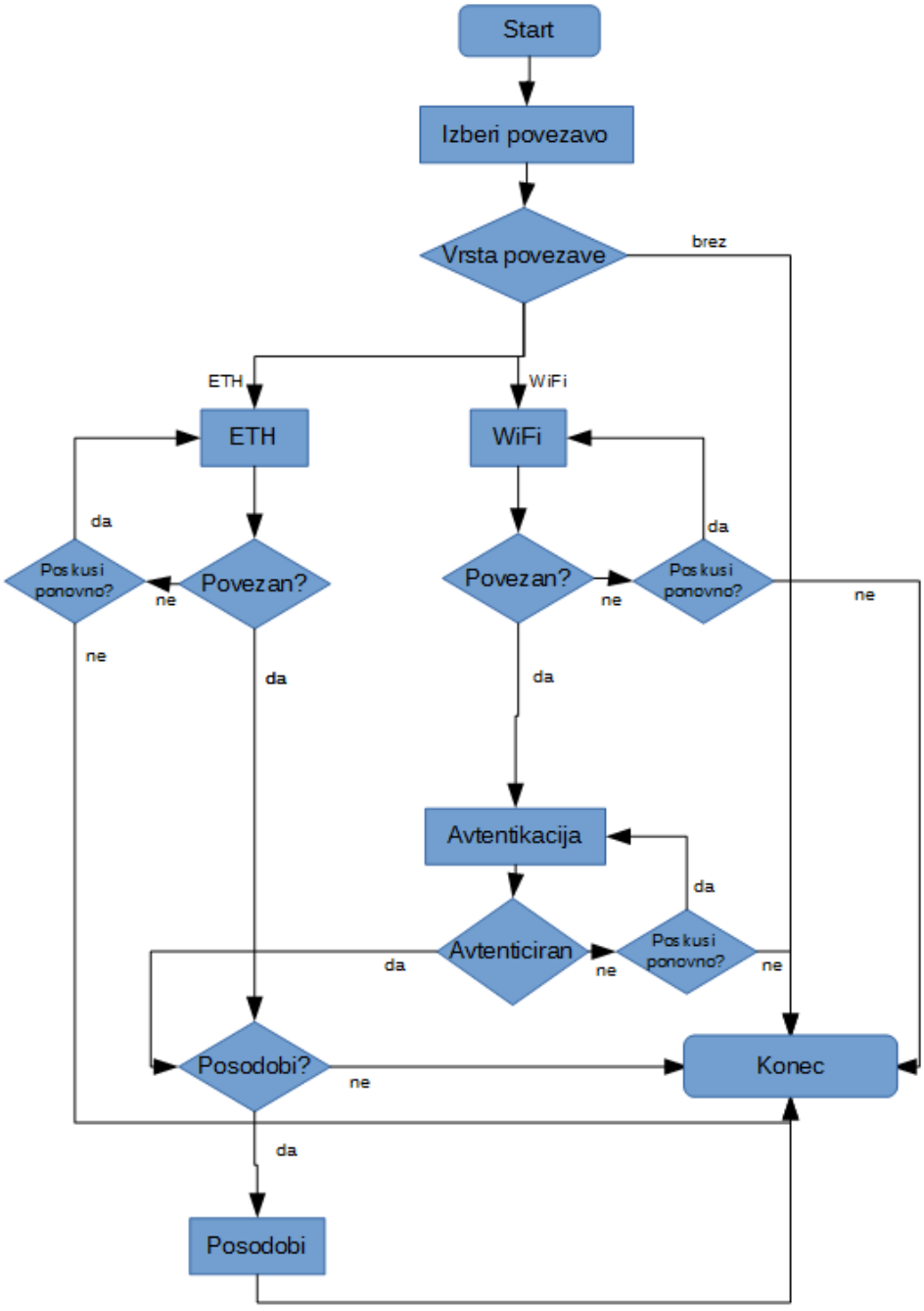
Pri vsakem ukazu se lahko namesto pozitivnega odgovora OK pojavi ERROR, v tem primeru se priporoča ponastavitev modula. Kot možnost izbire povezave na določeno dostopno točko smo implementirali kodo za vnos SSID-ja in pripadajočega gesla preko LCD zaslona, občutljivega na dotik. Implementacije smo se lotili tako, da smo ob pomoči funkcije drawButton() ustvarili prikaz z mnogo gumbi, ki predstavljajo posamezne znake (Slika 3.6). Uporabnik s klikom na posamezen znak tvori niz, ki se uporabi za dostop do AP. Večja pomanjkljivost obeh načinov povezovanja je nezmožnost vzpostavljanja povezave HTTPS zaradi premajhne moči procesorja. Izostanek slednje nam onemogoči varno avtentikacijo preko katere bi se lahko povezali s strežnikom. Za ohranjanje varnosti avtentikacijskih podatkov, ki jih uporabljamo za vstop v aplikacijo, smo kot alternativno avtentikacijo v aplikacijo vključili t.i. API ključ, ki pripada določenemu sistemu (entiteta v bazi, kar bo natančneje opisano v nadaljevanju) in se generira sočasno z ustvarjanjem le-tega. Uporabnik mora pri vsaki zahtevi HTTP kot parameter priložiti svoj API ključ, ki je hkrati identifikator sistema. Na ta način se lahko do neke

Slika 3.6: Vnos SSID

mere prepričamo, da gre za lastnika sistema in preprečimo, da bi napadalec prišel do uporabniškega imena in gesla, ki sta namenjena uporabi v spletni aplikaciji, v kolikor bi ju uporabljali za avtentikacijo brez varnosti.
Povezava v omrežje je ponujena tekom inicializacije MCU-ja ali kasneje kot izbira v meniju, ki je prikazan na LCD zaslonu. Diagram poteka na Sliki 3.7 prikazuje možnosti izbire povezave ob inicializaciji MCUja. Na primerih A.1 in A.2 si lahko pogledamo razliko v implementaciji generiranja zahtevka HTTP v odvisnosti od vrste povezave, tj. etherneta ali WiFi.

3.1.5 Senzor za temperaturo in vlago zraka

Za zaznavanje temperature in vlage zraka smo izbrali digitalni senzor DHT22, ki je zanesljiv, natančen in preprost za uporabo s pomočjo DHT knjižnice. Za delovanje potrebuje priklop na 5 V, GND in podatkovno linijo, ki je povezana z MCU. Obvezni element pri vezavi je tudi t. i. dvižni upor (Dvižni upor 3.1.11). Branje podatkov s senzorja je preprosto:



Slika 3.7: Diagram poteka za povezovanje z omrežjem in posodobitev

```
float temp = dht.readTemperature();
float humidity = dht.readHumidity();
```

Primer 3.5: Branje vrednosti DHT22

3.1.6 Vodoodporni senzor za temperaturo

Za merjenje vodne temperature smo izbrali digitalni senzor DS18B20, ki se nahaja v vodoodpornem ohišju.
Na eno vodilo, ki na MCU zasede zgolj en vhod, lahko priključimo več takšnih senzorjev, kar je mogoče zaradi unikatnega notranjega 64-bitnega naslova, preko katerega senzorje med seboj ločimo. Zgolj unikatni naslov ni dovolj za tovrstno vezavo, pomagamo si z OneWire knjižnico, ki v tem primeru skrbi za komunikacijo. Ker naslovi senzorjev niso fizično nikjer zabeleženi, smo si za iskanje le-teh pomagali z omenjeno knjižnico.

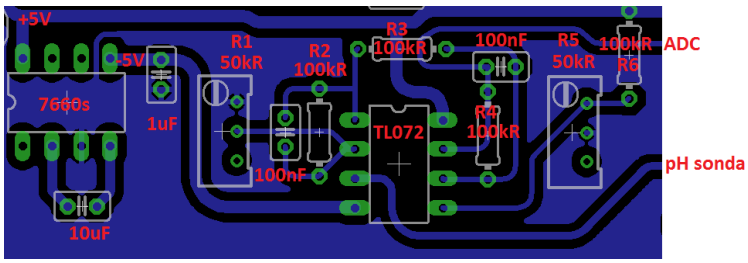
Ko so naslovi znani, jih lahko uporabimo na sledeč način:

```
DeviceAddress Probe02 = { 0x28, 0xCA, 0x35, 0xE6, 0x05, 0x00, 0x00, 0x6D
    };
  sensors.begin();
  sensors.requestTemperatures();
  float p2 = sensors.getTempC(Probe02);
```

Primer 3.6: Uporaba naslovov

3.1.7 pH

Meritve pH izvajamo s standardno kombinirano sondo z občutljivo stekleno membrano, za katero smo se odločili, ker v kombinaciji s pripadajočim vezjem omogoča kontinuirano beleženje vrednosti in je ob pravilnem rokovanju natančna in zanesljiva. Bolj kot izbor sonde, katere cena lahko sega od 5 € pa tudi preko 300 €, je pomembno vzdrževanje sonde, kamor sodi čiščenje in kalibracija, ki se izvaja programsko. Tako je lahko razlika med sondama v natančnosti, katerih razlika v ceni je desetkratna, zanemarljiva za potrebe v



Slika 3.8: pH vezje

akvaponiki, cenejša sonda pa bolj verjetno potrebuje gostejše intervale kalibriranja. Zato smo implementirali funkcijo (Primer A.3), ki skrbi za kalibracijo pH sond, da je izvajanje meritve karseda hitro in nenaporno, poleg tega pa ne potrebuje preprogramiranja mikrokrmilnika.

Na podlagi opisa (2.1.1), smo z namenom nižanja stroškov izdelali vezje, ki predstavlja vmesni element med sondo in MCUjem. Namen vezja je, da "okrepi" signal, za kar v glavni vlogi poskrbi operativni ojačevalnik (ang. op-amp) TL072. Vezje (Slika 3.8) je tiskano, na njem se poleg omenjenega nahaja še pretvornik napetosti 7660s, preko katerega dobimo -5 V, potrebnih za ojačevalnik, nekaj fiksnih uporov in kondenzatorjev, ki skrbijo za konstanten signal, ter 2 potenciometra, preko katerih nastavljamo zamik in ojačenje (ang. gain) na operativnem ojačevalniku. Poenostavljeno, ker sonda sama po sebi teoretično pri pH 0 generira napetost -0,414 V, z ojačenjem faktorja 5 dobimo napetost cca. -2,1 V, če temu prilagodimo še zamik, ki naj bo 2,1 V, pri pH 0 z mikrokontrolerjem izmerimo 0 V, pri pH 14 pa 4.2 V, tako so meritve precej bolj natančne, kot če bi merili nižje napetosti, poleg tega navadno merimo pH zgolj na določenem razponu npr. 6-9 in posledično prej omenjena parametra prilagodimo primeru. To pa ni edini razlog za uporabo vezja, operativni ojačevalnik namreč vsebuje tudi druge lastnosti, ki so ključnega pomena za ustrezno branje vrednosti, a se v tem diplomskem delu v to ne bomo poglabljali.

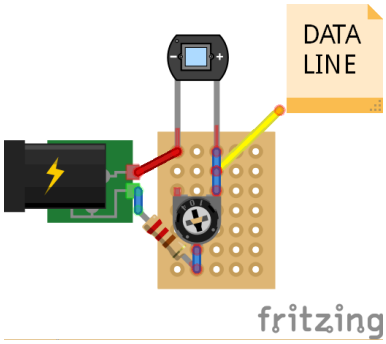
3.1.8 PAR senzor

Senzor za fotosintetsko aktivno sevanje (ang. Photosynthetically Active Radiation), mora biti občutljiv na svetlobo, ki spada v fotosintetični aktivni spekter, tj. na razponu 400-700 nm valovne dolžine elektromagnetnega valovanja in mora biti približek idealnemu kvantnenu odzivu (ang. ideal quantum response). Prav tako mora biti senzor vodoodporen in neobčutljiv na ribje dotike [12]. Ker je izvirni PAR senzor zelo drag, smo za potrebe pridobivanja PAR vrednosti izdelali lastni senzor, ki dobro posnema lastnosti profesionalnega. Oprema za izdelavo (Slika 3.9):

- fotodioda VTB8441BH;
- pleksi (50-60% prepustnost svetlobe);
- fiksni in nastavljivi upor oz. potenciometer;
- vodotesno plastično ohišje in uvodnica;
- UTP kabel in AUX konektor.

Fotodiodo smo izbrali na podlagi lastnosti, ki so razvidne iz podatkovne tabele. Njeno spektralno območje sega od 330 nm do 720 nm, kar pomeni, da zajema celotno območje PAR meritev. Značilnost fotodiode je, da proizvede tok, ki je približno sorazmeren moči sevanja svetlobe. Fotodiodo smo v tokokrog (Slika 3.10) vezali v t. i. reverse-biased načinu. V tem primeru lahko, ko na diodo pade svetloba, na strani anode izmerimo pozitivno napetost, ki je odvisna od moči svetlobe oz. v našem primeru vrednosti PAR. Zaradi omejenih sposobnosti strojne opreme oz. mikrokrmilnika smo vezani na tovrsten način delovanja, mikrokrmilnik nam namreč omogoča branje napetosti preko analognega vhoda. Mega2560 privzeto uporablja 5 V referenčno napetost, kar pomeni, da pri 10-bitni natančnosti ADC pretvornika preberemo razliko v napetosti velikosti 0,0049 V ali 4.9 mV. Senzor smo kalibrirali s poskušanjem in primerjanjem vrednosti s tistimi, ki jih daje profesionalni. Kot začetno točko smo privzeli nedokazano dejstvo, da PAR vrednost pri

Slika 3.9: PAR senzor



Slika 3.10: PAR vezje

neposredni sončni svetlobi znaša približno $2000\mu\text{mol}m^{-2}s^{-1}$ in ugotovili, da z majhnimi premiki potenciometra ter množenjem prebrane povprečene (100 branj) vrednosti z 1,9 vrne pričakovan rezultat. Od tu dalje smo preizkušali senzor pri različnih osvetlitvah in prostorih in ves čas vrednosti primerjali s tistimi na izvirnem senzorju. Po izčrpnem testiranju smo ugotovili, da je pri prebranih vrednostih do 500 ustrezen faktor 1,97, pri vrednostih nad 500 pa 1,91. Skupni upor smo nastavili na 19 kOhm. Primer kode, ki skrbi za izračun par vrednosti, je podan v Primeru A.4.

3.1.9 Peristaltične črpalke

Za dovajanje tekočin, ki uravnavajo pH, uporabljamo peristaltične črpalke, kjer se tekočina ne dotika nobenih mehanskih delov, deluje namreč na principu podtlaka, s pomočjo katerega premika tekočino vzdolž cevi. Črpalke za obratovanje potrebujejo napetost 12 V in relativno veliko toka, oboje zagotavljamo s pomočjo zunanjega napajanja (3.1.14). Ker črpalke prožimo z ukazom na MCUju in tega ne moremo početi neposredno, kot vmesni element uporabljamo rele.

Rele je električno nadzorovano stikalo. Uporablja se, kjer je s šibkim signalom potrebno nadzorovati močnejše naprave, hkrati poskrbi za delno električno izolacijo med napravami. [9]

Črpalko torej prožimo s signalom preko izhodno definirane nožice, kar lahko povzroči trenutno ali pa trajno škodo na MCUju. Ker je črpalka pravzaprav elektro motor, se ob njenem ugašanju pojavi t. i. električna špica, ki lahko presega napetosti 100 V. Da bi se temu izognili, smo se odločili za vezje, kjer je pred vsakim relejem nameščen optični izolator (čip, ang. optocoupler), kar pomeni, da je edina povezava med zunanjim tokokrogom in MCU infrardeči žarek.

Ker se specifike vsake od črpalk do neke mere razlikujejo, smo pred uporabo v sistemu vsaki izmerili pretok. Da bi bil slednji izmerjen karseda natančno, smo črpalke najprej namestili v končni položaj s pravilnimi dolžinami

cevi, nakar smo jih napolnili z vodo, da smo se znebili zraka in konec v smeri toka speljali v merilni valj. Posamezno črpalko smo vključili za čas 60 s in zabeležili pretočeno količino, ki predstavlja pretok na minuto. Ker črpalke s staranjem spreminjajo lastnosti ali pa spreminjamo njihovo postavitev, je potrebna vnovična kalibracija, zato smo implementirali kodo, ki uporabniku s pomočjo zaslona narekuje navodila in z vnesenimi parametri sama izračuna potrebno količino vnosa uravnalnih tekočin, kar je prikazano na s pomočjo diagrama na Sliki 3.11.

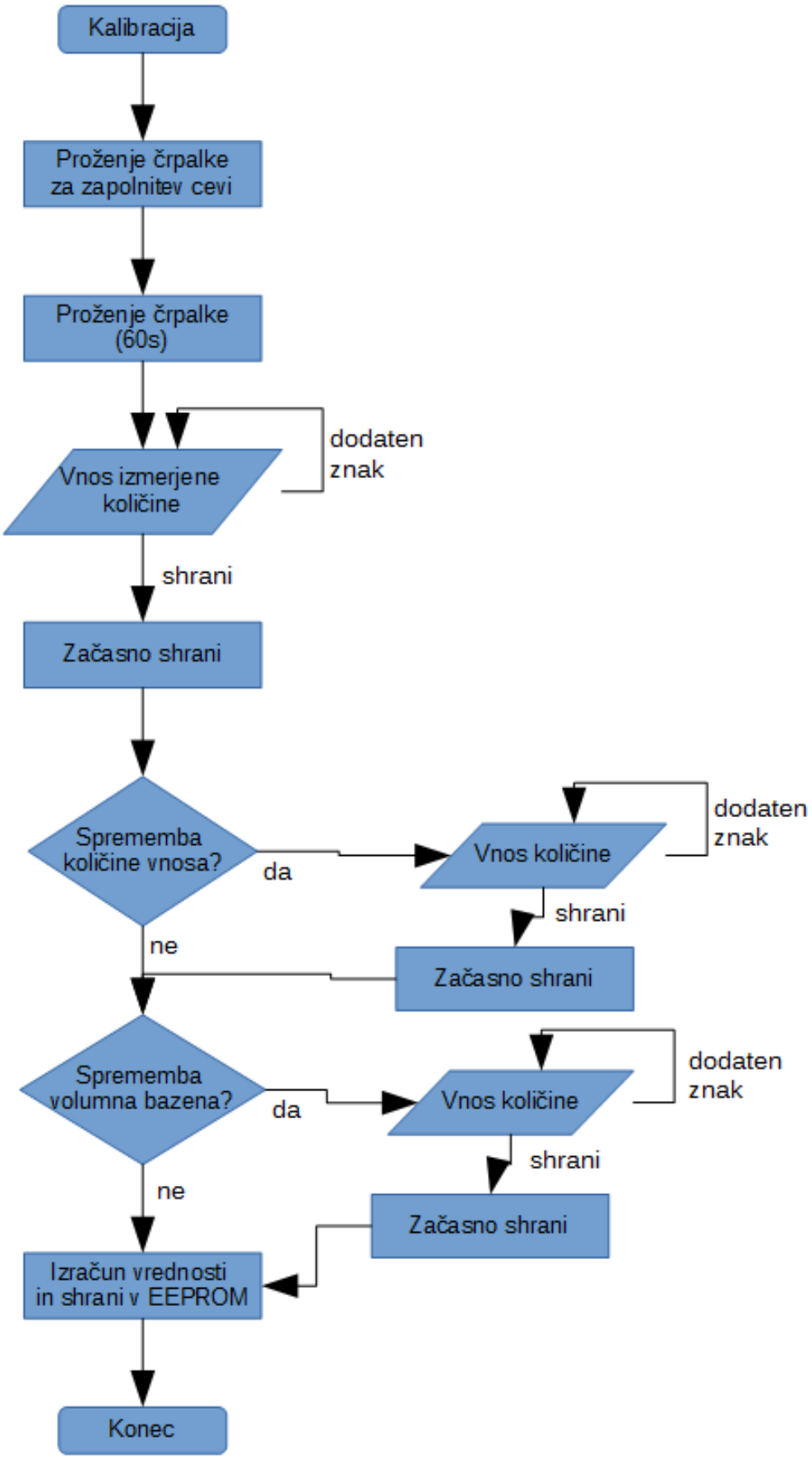
Na posamezen sistem potrebujemo dve črpalki za dodajanje dveh različnih uravnalnih tekočin, kar je pomembno predvsem zaradi rib, ki so občutljive na vnos raztopin in je zanje manj škodljivo, če izmenično dovajamo dve različni. V primeru, da pride do izpada električnega toka ali ponastavitve, je pomembno, da se vrstni red dovajanja ohrani, zato ob vsakem črpanju v EEPROM zapišemo, katera je na vrsti naslednja, to vrednost pa nato preberemo v funkciji setup().

```
int pump_turn = 0;
void setup()
{
        pinMode(30, OUTPUT);
        pump_turn = EEPROM.readInt(ADDR);
}
```

Primer 3.7: Branje vrednosti iz EEPROMa

EEPROM (ang. Electrically Erasable Programmable Read-Only Memory) je vrsta spomina, ki ohrani vsebino tudi ob izgubi energije. Lahko ga beremo, brišemo in vanj zapisujemo.

Možen je tudi primer, da uravnalnih tekočin zmanjka, v tem primeru je nesmiselno, da črpalke delujejo, zato pred vsakim črpanjem najprej preverimo stanje plovnega stikala. Pozorni moramo biti tudi na to, da med posameznimi črpanji tekočin v sistem mine dovolj časa, kar preverjamo s pomočjo urinega modula RTC DS3231. Slednji s pomočjo baterije ohrani informacije o trenutnem času tudi ob izpadu električne energije.



Slika 3.11: Diagram poteka za kalibriranje črpalk

Slika 3.12: Plovno stikalo

3.1.10 Avtomatski hranilnik

Za potrebe hranjenja rib smo razvili avtomatski hranilnik, čigar glavna komponenta je linearni aktuator. Ta skrbi za sproščanje odprtine, skozi katero pada hrana. Gre za servo motor, ki se med drugim uporablja pri centralnem zaklepanju v vozilih. Njegova funkcija je omejen hod vodila naprej in nazaj. Proži se ga podobno kot peristaltični črpalki, preko releja. Aktivacija hranilnika potega preko LCD zaslona, s klikom na gumb ali periodično, s pomočjo urinega modula.

3.1.11 Plovno stikalo

Za potrebe nadzora prisotnosti dozirnih tekočin smo v posamezno posodo, kjer je tekočina shranjena, dodali magnetno plovno stikalo. Za lažjo predstavo, kakšno je videti plovno stikalo, si lahko bralec ogleda Sliko 3.12.

Če bi pogledali v notranjost stikala, bi našli dve nepovezani žici, ki sta si relativno blizu. Čeprav žici v svojem naravnem položaju nista staknjeni, se to spremeni, ko spreminjamo položaj plovca, na katerem se nahaja magnet. Kadar žici nista staknjeni, pravimo, da je stikalo odprto in nasprotno, ka-

dar sta povezani, pravimo, da je stikalo zaprto. Kdaj je v določenem stanju glede na položaj plovca, se razlikuje od naprave do naprave. V konkretnem primeru je odprto, ko se plovec približa koncu oz. zatiču. To je pomembno za pravilno namestitev naprave v okolje, kjer bo obratovala, čeprav lahko z negacijo v programski kodi dosežemo želen učinek ne glede na orientacijo plovca.

Na Arduino razvojni ploščici je za vsako digitalno nožico na voljo t. i. notranji dvižni upor. Omogočimo ga tekom postavitve s preprostim ukazom

```
pinMode(FLOAT_SWITCH_PIN, INPUT_PULLUP);
```

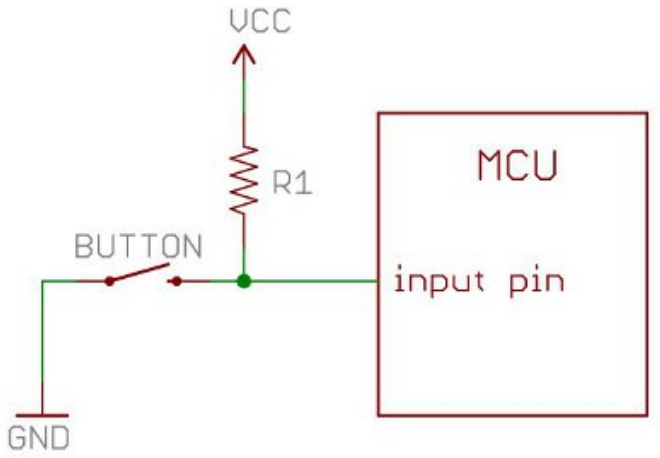
Primer 3.8: Omogočanje notranjega dvižnega upora

Dvižni upori (ang. pull-up resistor) so zelo pogosti pri uporabi mikrokrmilnikov oz. pri ostalih logičnih napravah. Če določeno nožico definiramo kot vhod in nanjo ne priklopimo ničesar, ne moremo z gotovostjo trditi, kakšno stanje zaznava. Če nismo prepričani ali je neka nožica v visokem ali nizkem stanju, pravimo, da plava (ang. floating). Da preprečimo plavajoče stanje, v krog vključimo dvižni (ang. pull-up) upor. V takem primeru imamo na vhodu vedno nizko ali visoko stanje. Pri dvižnem uporu je stanje visoko, kadar naprava, s katero operiramo, ni povezana, kar lahko vidimo na Sliki 3.13. Notranji dvižni upor deluje na isti način kot zunanji na sliki. Če bi tovrsten upor izpustili in na vhod neposredno pripeljali napajanje, bi ob vklopu lahko prišlo do kratkega stika.

Kadar želimo programsko preveriti, ali neka naprava operira, to naredimo tako, da preverimo, če je stanje na pinu nizko.

```
if (digitalRead(FLOAT_SWITCH_PIN) == LOW) {
        // circuit closed
}
```

Primer 3.9: Branje stanja na vhodu



Slika 3.13: Dvižni upor

Na tovrsten način preverjamo prisotnost tekočine pred doziranjem. Na nožico, kjer se nahaja plovno stikalo, smo povezali tudi prekinitev (ang. interrupt). Prekinitev je poseben dogodek, ki ga mikrokrmilnik obravnava takoj. V primeru Arduino Mega 2560 razvojne ploščice smo omejeni na izbiro 6 pinov, le toliko jih namreč podpira prekinitve. Iz podatkovne tabele je razvidno, da se prekinitev z indeksom 4 nahaja na nožici z oznako 19. Na nožici 19 smo zato vključili dvižni upor in dodali prekinitev, ki se proži, kadar se stanje spremeni iz visokega v nizkega, to določa parameter FALLING.

```
volatile long lastSwitch = 0;

void setup() {
  pinMode(19, INPUT_PULLUP); // enable pull-up
  attachInterrupt(4, warning, FALLING);
}
void warning() {
        long now = millis();
        if (now - lastSwitch < 300) {
                sendMail();
        }
        lastSwitch = now;
}
```

Primer 3.10: Povezava prekinitve in ignoriranje pojava odbijanja

V trenutku, ko stikalo sklene krog, se v resnici to lahko zgodi večkrat zaporedoma v izjemno kratkem času, čemur pravimo odbijanje(ang. bounce) [9]. Prekinitev se vsakič ponovno proži, a kar se izvede znotraj prekinitve, lahko preprečimo tako, da slednjo preprosto ignoriramo, če ni pretekel čas, v katerem se je odbijanje z veliko gotovostjo prenehalo.

3.1.12 Senzor gibanja

Za ohranjanje zaslona smo dodali senzor gibanja HC-SR501, ki skrbi, da se zaslon aktivira le takrat, ko se mu nekdo približa. Gre za celosten modul, ki za delovanje potrebuje le priklop na napajanje in povezavo z MCU za branje signala. Kadar senzor zaznava gibanje, na izhod, ki je povezan na MCU vhod, pošlje visok signal, ki ga obdrži toliko časa kot znaša čas zaklepa. Po pretečenem času se stanje spremeni v nizkega. Da se ekran odzove karseda hitro, smo na MCU izbrali vhodno nožico, ki podpira prekinitve. Implementacija je podobna kot pri plovnem stikalu, s to razliko, da nas tokrat zanimata oba prehoda stanj iz visokega v nizko in iz nizkega v visoko stanje. Ker sta nožici 2 in 3 zasedeni s strani LCD zaslona, nožici 20 in 21 s strani urinega modula in nožica 19 s strani plovnega stikala izberemo nožico 18, zadnjo preostalo, ki podpira prekinitve.

```
int state = 0;
void setup() {
  pinMode(18, INPUT);
  attachInterrupt(5, lcdOnOff(), CHANGE);
}
void lcdOnOff() {
  state = digitalRead(MOTION_PIN);
  if (state == HIGH)
  {
        LCD.lcdOn();
  }
  else if (state == LOW)
  {
    LCD.lcdOff();
  }
}
```

Primer 3.11: Zaznavanje gibanja

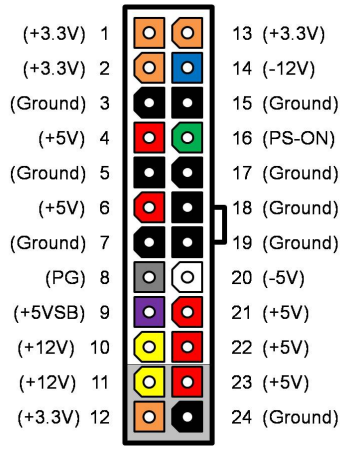
Slika 3.14: Kalibracija preko zaslona

3.1.13 Kalibracija

Kalibracija se lahko izvaja na vseh poljubnih parametrih. Za razliko od kalibracije, specifične za pH, smo pri tej omogočili prirejanje vrednosti v smislu prištevanja in odštevanja. Če npr. temperatura zraka odstopa za vrednost 1,5 v minus, ji enako vrednost prištejemo. To lahko storimo preko spletne aplikacije, nakar preko povezave z omrežjem posodobimo tudi vrednosti na MCUju ali pa kalibriramo vrednosti s pomočjo menija, prikazanega na LCD zaslonu (Slika 3.14) in v nasprotni smeri vrednosti posodobimo na spletni aplikaciji.

3.1.14 Napajalnik

Ker mnogo vključenih komponent potrebuje ločeno napajanje, smo v rešitev dodali univerzalni napajalnik, ki se navadno nahaja v računalniških ohišjih. ATX napajalnik se je izkazal za primerno rešitev, ker vključuje mnogo različnih napetostnih nivojev (Slika 3.15)



Slika 3.15: Napetosti na nožicah

- 3.3V za WiFi modul
- 5V za pH vezje
- 12V za Arduino, hranilnik in črpalke

in premore močan električni tok, ki je nujen za poganjanje avtomatskega hranilnika in črpalk. Na konec žic smo prilotali DC 2,1 mm priključke, preko katerih komponentam dovajamo želene napetosti. Za vklop napajalnika smo pin PS-ON preko stikala povezali z Ground pinom.

3.2 Strežnik

3.2.1 Ruby

Ruby je dinamičen, popolnoma objektno-orientiran programski jezik za splošno rabo. Uporablja dinamičen sistem za določanje tipov in sam upravlja s pomnilnikom. Uporablja svoj paketni upravljalni sistem (ang. packet management system), ki mu pravimo RubyGems, podobno posameznem modulu rečemo gem. Slednji so na voljo v velikem številu in tako razvijalcu olajšajo delo. [7]

3.2.2 Rails

Rails je ogrodje, zgrajeno na Ruby programskem jeziku. Hkrati je domensko-specifični jezik za hiter razvoj spletnih aplikacij, ki zagovarja koncept "convention over configuration", kar pomeni, da čim manj odločitev za konfiguracijo ostane programerju. Rails sledi MVC arhitekturi, pri tem mu pomaga REST implementacija usmerjanja. Sledi tudi principu DRY. [8]

3.2.3 PostgreSQL

PostgreSQL je odprtokodna objektno-relacijska baza za shranjevanje podatkov, ki jo je moč uporabiti na različnih platformah. Je tudi ena najnaprednejših in najpogosteje uporabljenih baz v produkcijskih okoljih. [5]

3.2.4 Heroku

Heroku je oblačna platforma za razvoj in gostitve aplikacij. Prispeva k hitremu razvoju, saj omogoča neprestano vnovično nalaganje aplikacije na strežnik. Razvijalcu pomaga, da se ta osredotoči na razvoj in se ne ukvarja z upravljanjem strežnika in vzdrževanjem stanja. Podpira integracijo s sistemi za nadzor različic kot je Git. [10]

Poglavje 4

Razvoj in opis aplikacije za nadzor in oddaljeno upravljanje

Glavna ideja za razvoj aplikacije izhaja iz opisa motivov (2.1.2). Aplikacija je namenjena uporabnikom, ki želijo imeti nadzor nad sistemom in lahko upravljajo s parametri, kot tudi tistim, ki želijo pregled nad vrednostmi posameznih parametrov v grafični in tabelarični obliki. Kljub temu, da je aplikacija v prvotnem načrtu razvita za potrebe nadzora akvaponskih sistemov, zaradi svoje zasnove ni omejena zgolj na tovrstne primere, pač pa je primerna tudi za marsikateri drugi projekt, za katerega želimo, da vsebuje beleženje podatkov oz. omejen nadzor nad parametri in strojno opremo.

4.1 Razvoj in opis aplikacije

Za dostop do aplikacije je sprva potrebna registracija - ta poteka preko pojavnega okna, ki se odpre s klikom na povezavo "Create an account" ob podpori AJAXa. Uporabnik mora v obrazec vnesti ime, naslov spletne pošte in geslo, s pomočjo slednjih dveh tudi dostopa do aplikacije. Po uspešno opravljeni registraciji se vanjo lahko vpiše in s tem pridobi dostop za grajenje nadzorne plošče, ki je sestavljena iz več nivojev. Ob vpisu je mogoče izbrati možnost ohranjanja seje ob vseh nadaljnjih dostopih, kar je implementirano s pomočjo

piškotkov (ang. cookie).

Piškotki se uporabljajo za shranjevanje manjših količin podatkov v za to namenjen prostor brskalnika. Podatki nastopajo v obliki ime/vrednost in so namenjeni predvsem ohranjanju nekega stanja (ang. stateful resource). [4]

Ob posredovanju avtentikacijskih podatkov strežnik s poizvedbo v bazo ugotovi pravilnost slednjih in hkrati ugotovi, ali gre za navadnega uporabnika oz. administratorja, na podlagi česar generira pogled in kasneje preverja pravice za dostop do implementiranih akcij, tj. brisanje entitet, nastavljanje kalibracij ... Ločeno na nivoju krmilnika (ang. controller, Rails izraz za komponento) preverjamo pravice za dostop do vsebine aplikacije, ki je praviloma na voljo le lastniku slednje, sicer bi do nje lahko dostopali vsi.

```
before_action :this_user, only: [:show, :destroy]
    def this_user
      @proj = curr_user.projects.find_by(id: params[:id])
      if @proj.nil?
        redirect_to root_url
      end
    end
```

Primer 4.1: Preverjanje dostopa za uporabnika

V danem primeru se preverja, če je uporabnik, ki želi dostopati do prikaza projekta, lastnik slednjega, v kolikor ni je preusmerjen na začetno stran.

Po vpisu v aplikacijo je uporabnik preusmerjen na začetno stran, ki predstavlja seznam ustvarjenih projektov. Začetno stran definiramo v datoteki routes.rb z ukazom

```
root 'projects#index'
```

Primer 4.2: Definiranje začetnega pogleda

strežnik tako ve, da se koda za prikaz nahaja v krmilniku Projects (app/controllers/projects_controller.rb) pod akcijo index.

```
class ProjectsController < ApplicationController
  def index
    unless current_user.nil?
      @user = current_user
      @projects = @user.projects
    end
  end
end
```

Primer 4.3: Krmilnik Projects in akcija index

Ob koncu akcije sledi generiranje istoimenskega pogleda, ki smo ga definirali v datoteki index.html.erb, slednja se nahaja v določeni datotečni strukturi /app/views/projects/.

V pogled smo vključili t. i. delce (ang. partial), kot jim pravimo v Rails žargonu. Tako se držimo principa DRY, ki ga priporoča Rails, saj lahko isto kodo delcev vstavimo na več mestih v aplikaciji.

```
<% if @projects.any? %>
  <div class="row">
    <%= render @projects %>
  </div>
<% end %>
```

Primer 4.4: Prikaz delcev

Na tem primeru lahko vidimo, da je spremenljivka @projects, ki smo jo definirali v projects_controller.rb, dostopna v index.html.erb, kjer se generira pogled. V konkretnem primeru se preverja, ali je v spremenljivki shranjen kakšen projekt s pomočjo funkcije .any?, v kolikor je, ukazu "render @projects" sledi prikaz posameznih projektov, katerih pogled je definiran v t. i. "partial" datoteki. Delce predstavlja datoteka, ki se začne z "_", v danem primeru torej _project.html.erb (Primer B.1).

Ko se uporabnik nahaja na strani s projekti, jih lahko poljubno pregleduje, briše, ustvarja. Vse akcije dodajanja, urejanja in brisanja smo implementirali preko t. i. pojavnih obrazcev, pri čemer smo si pomagali z

možnostjo remote: true, ki jo podamo kot parameter v link_to generator povezav. S slednjo opcijo smo onemogočili privzeto potrjevanje, ki ga izvaja brskalnik in posledično sami implementirali kodo, ki poskrbi za AJAX klic. Ker smo na celotni aplikaciji skoraj vse povezave in obrazce implementirali na isti način in se držimo principa DRY, smo vrhnje plasti obrazca definirali le na pogledu, ki je skupen vsem ostalim, tj. v aplication.html.erb. Ko želimo kasneje prikazati določen obrazec, to storimo preko prej omenjenih delcev, ki smo jih predhodno definirali, da so skladni s posameznim modelom iz baze. Na primeru projekta poglejmo, kako poteka celoten proces od definiranja novega modela, do njegovega zapisa v bazo.

```
<%= link_to "+Add project", new_project_path, remote: true, class: "
    button button-border-primary button-rounded right" %>
```

Primer 4.5: Ustvarjanje povezave s pomočjo UrlHelperja

Drugi parameter funkcije link_to sprejme simbol za url povezavo, ki je avtomatično generiran s tem, ko smo v routes.rb kot RESTful vir definirali projekt

```
resources :projects, only: [:show, :new, :destroy, :create]
```

Primer 4.6: Definiranje RESTful vira

Ko kliknemo na povezavo, generirano z link_,to nas new_project_path preusmeri na naslov .../projects/new, ki ga strežnik avtomatično poveže s krmilnikom Projects in akcijo new, znotraj katere se nahaja generiranje novega praznega modela projekta, ki služi pri formiranju obrazca.

```
class ProjectsController < ApplicationController
  def new
    @project = Project.new
  end
end
```

Primer 4.7: Definiranje praznega modela

Ker izvajamo ajax klic, strežnik po končani akciji izvede javascript kodo v istoimenski datoteki, tj. new.js.erb,

```
$('#dialog h3').html("<i class='glyphicon glyphicon-plus'></i> Add New
    Project");
$('.modal-body').html('<%= j render("form") %>');
$('#dialog').modal("show");
```

Primer 4.8: JS koda za prikaz dialoga

kjer dialogu podamo ime, vanj prenesemo obrazec (ang. form) in ga prikažemo kot pojavno okno. Vse obrazce na strani smo implementilari s pomočjo funkcije form_for, slednja je del Railsa in kot parameter sprejme model, bodisi novo ustvarjen bodisi obstoječ, s katerim želimo manipulirati. Primer obrazca za projekt, ki se med različnimi modeli razlikuje zgolj v odvisnosti od atributov, je viden v Primeru B.2.

Na tem mestu velja omeniti, da smo tabele oz. modele v bazi generirali s pomočjo terminala (operacijski sistem Linux), za primer projekta je ukaz videti

```
rails generate model Project name:string user:references
```

Primer 4.9: Ukaz za generiranje novega modela

pri tem je pomembno, da se nahajamo znotraj mape, ki predstavlja Rails projekt. Ukazu podamo parametre <ime tabele/modela><stolpec:tip>, na podanem primeru opazimo posebnost :references, slednja namreč ne predstavlja tipa, pač pa povezavo med tabelama Project in User, ki sta medsebojno povezani preko tujega ključa. Ob izvedbi ukaza, se v /app/db/migrate/ generira datoteka, ki skrbi za izvedbo t. i. migracije preko ukazov, ki jih razume in izvede Rails. Vzajemno se generira tudi datoteka /app/models/project.rb, kjer so naknadno navedene vse omejitve, lastnosti in povezave modela.

Po vpisu podatkov, ki zadevajo projekt, Rails v ozadju preveri, če gre za nov oz. posodobljen objekt in se na podlagi tega odloči, kateri akciji bo predal podatke s potrjenega obrazca. Ker tokrat izhajamo iz novo ustvarjenega modela, sledi akcija create, ki ga shrani v bazo, pripadajoča JS koda pa poskrbi za zaprtje obrazca.

Če rečemo, da smo nadzorno ploščo implementirali v več plasteh, model Project predstavlja najvišjo. Na naslednji se nahaja model System, ki si ga lahko zamislimo kot posamezno enoto, ki jo nadziramo, to je lahko bodisi bazen bodisi gojišče ali pa nekaj poljubnega. Sistem pripada projektu, slednji jih lahko vsebuje poljubno mnogo. Na zadnjem nivoju se nahajajo parametri – ti pripadajo sistemu, ki jih lahko vsebuje največ 9, tovrsten pogoj pa izhaja iz omejitev pri prikazu podatkov. Vsakršno preverjanje omejitev smo implementirali na modelu s pomočjo vgrajenih pomagal (ang. helperjev), alternativna možnost bi bila ročno preverjanje v krmilnikih pripadajočih modelov. Preverjanje za zgornji primer smo implementirali na sledeč način:

1. modelu System smo dodali atribut parameters_count;
2. v parameters.rb smo dodali
   validates_associated :system in
   belongs_to :system, counter_cache: true, slednji skrbi za avtomatsko inkrementiranje vrednosti v prej definiranem atributu parameters_count;
3. v system.rb smo dodali validates :parameters_count, numericality: { less_than: 9 }

kar pomeni, da se ob vsakem dodanem parametru na sistem, slednjemu atribut parameters_count poveča za 1. Ob dodajanju parametra se izvede tudi validacija za sistem (korak 2) – če je ta neuspešna, novega parametra ni moč ustvariti.

Sistem je bolj kompleksen kot projekt zaradi nadaljnjih asociacij z ostalimi modeli, ki jih vsebuje, kot je razvidno iz sheme baze (Slika 4.5). Na en sistem je predvidena uporaba enega MCUja, ki s spletno aplikacijo komunicira preko API ključa, slednji služi tudi kot identifikator sistema.

Za čim lažji in učinkovit nadzor je z določenimi parametri mogoče upravljati preko zaslona, občutljivega na dotik, in hkrati preko uporabniškega vmesnika, ki ga zagotavlja spletna aplikacija. Komponenti sta povezani preko APIja. Ko uporabnik ustvari sistem, ki predstavlja skupek parametrov, ki so medsebojno povezani, strežnik generira ključ za dostop, ki zagotavlja nizko

stopnjo varnosti, da lahko uporabnik na strežnik pošilja podatke. Takemu ključu pravimo API ključ (ang. API key). V danem primeru je uporabljen algoritem SHA1 dolžine 160-bitov, ki služi kot zgoščevalna funkcija.

Zgoščevalna (ang. hash) funkcija je tista, ki preslika neko vrednost kakršne koli dolžine v zgoščeno vrednost fiksne dolžine. Med dobre sodijo tiste, katerih zgoščena vrednost se malokdaj ponovi, na kar neposredno vpliva tudi njena dolžina.

Ključ se generira na podlagi podanega niza, ki je sklopljen iz e-pošte uporabnika in časa ustvarjenja sistema (Primer B.4). Tako je niz zagotovo unikaten, ker lahko nek uporabnik v danem trenutku ustvari le en sistem. Kljub temu, da obstaja možnost, da bi zgoščevalna funkcija za dva različna niza vrnila enako zgoščeno vrednost, je verjetnost zanemarljivo majhna.

Parametri, ki so del sistema, so tisti, katerih vrednosti spremljamo. Slednje se shranjujejo v vmesno tabelo Entries, kjer posamezen vpis zaznamuje vrednost določenega parametra, ki pripada izbranemu sistemu. Da lahko določimo, katere vrednosti časovno sodijo skupaj, kar pomeni, da so rezultat enega pošiljanja, smo vpisom določili atribut set (množica). Za zapis vrednosti uporabljamo API:

.../entries/save_multiple

POST, ustvari zapise

oblika zahtevka

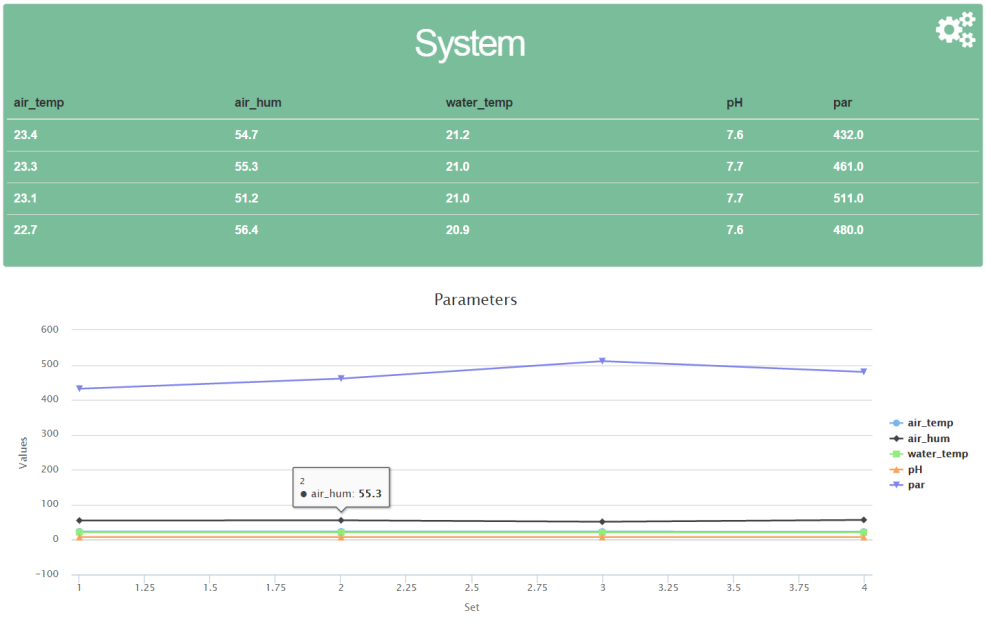
application/json

```
{
    "api_key": "115f64ee8ea246ac8251d8453798094bac6207e0",
    "entries": [
        { "name": "ph", "value": "7.2"},
        { "name": "air_temp", "value": "25"}
    ]
}
```

oblika odgovora

200 - aplication/json

```
{
    "status" : "ok"
}
```



Slika 4.1: Prikaz vrednosti

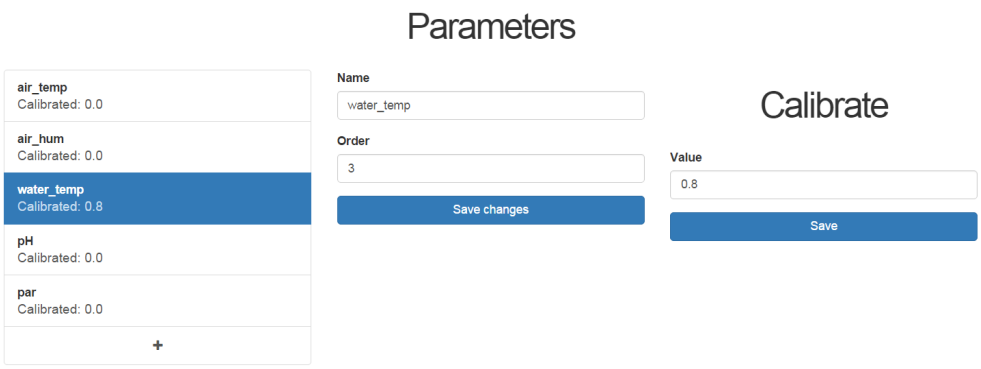
Ko so vrednosti zabeležene, si jih lahko ogledamo tabelarično in grafično (Slika 4.1). Tabele smo tako kot velik del preostale aplikacije implementirali s pomočjo bootstrap knjižnjice in pripadajočih predefiniranih html razredov. Graf, ki se nahaja spodaj, je implementiran s pomočjo knjižnice highcharts (Primer 5.1).

Poleg shranjevanja vrednosti smo v aplikacijo dodali tudi podporo za kalibriranje parametrov (Slika 4.2).

Ob ustvarjanju novega sistema se izvede funkcija, ki shrani nov vnos v tabelo calibration_version, slednji sestoji iz trenutne verzije kalibracije in identifikatorja sistema.

```
after_create :create_associated_calibration_version
  def create_associated_calibration_version
        self.build_calibration_version(version:0)
        self.save
  end
```

Primer 4.10: Ustvari verzijo kalibracije



Slika 4.2: Kalibracija parametrov

API za trenutno verzijo kalibracije:

.../systems/cal_ver

POST, trenutna verzija

oblika zahtevka

application/json

```
{
    "api_key": "115f64ee8ea246ac8251d8453798094bac6207e0",
}
```

oblika odgovora

200 - aplication/json

```
{
    "version" : "6"
}
```

Podatek o trenutni verziji beležimo, ker tako na MCUju primerjamo trenutno nameščeno s tisto, ki se nahaja na spletu. Če se razlikujeta, lahko zahtevamo posodobitev kalibracijskih podatkov, kar poteka preko APIja:

.../systems/calibration

POST, kalibracijski podatki

oblika zahtevka

application/json

```
{
    "api_key": "115f64ee8ea246ac8251d8453798094bac6207e0",
}
```

oblika odgovora

200 - aplication/json

```
-0: {
    id: 14
    name: "air_temp"
    order: 1
    -calibration: {
        value: 2.1
    }
}
```

Menjava verzije se proži ob vsaki uspešni posodobitvi kalibracije poljubnega parametra

```
@param.system.calibration_version.increment!(:version, 1)
```

Primer 4.11: Inkrementiranje verzije kalibracije

Kalibracija in posodobitev podatkov je možna tudi v drugi smeri, tj. od MCUja na aplikacijo.

.../calibrations/update_multiple

POST, ustvari kalibracijske zapise

oblika zahtevka

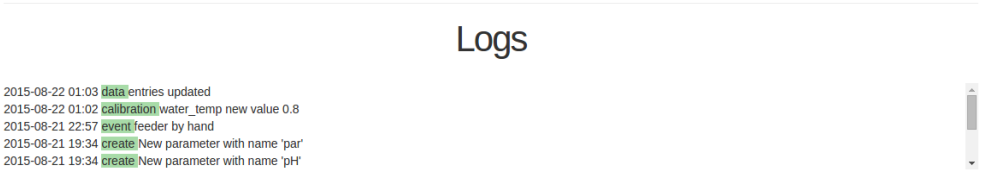
application/json

```
{
    "api_key": "115f64ee8ea246ac8251d8453798094bac6207e0",
    čalibrations": [
        { "name": "ph", "value": "0.2"},
        { "name": "air", "value": "0.4"}
    ]
}
```

oblika odgovora

200 - aplication/json

```
{
    "status" : "OK, calibrations saved"
}
```



Slika 4.3: Zapisi preteklih dogodkov

Zelo pomembno je tudi beleženje preteklih dogodkov. V ta namen smo definirali model Log, ki je namenjen shranjevanju zapisov, ki pričajo o preteklih akcijah. Nekatere informacije se shranjujejo ob npr. ustvarjanju novega parametra, kalibriranju, spremembi imena... Tovrstne zapise ustvarjamo v krmilnikih ob danih akcijah. Za informacije, ki nosijo večjo vrednost, je zadolžen MCU, ta namreč beleži, kdaj je v sistem dodal uravnalne tekočine, kdaj so bile nahranjene ribe, kdaj je nekdo fizično preko zaslona sprožil črpalke, ob morebitni izgubi povezave zabeleži ponovno vzpostavitev... Tako nam je na voljo zgodovina, ki vsebuje pomembne informacije (Slika 4.3). MCU zapise ustvarja preko APIja:

.../logs/create

POST, zabeleži dogodek

oblika zahtevka

application/json

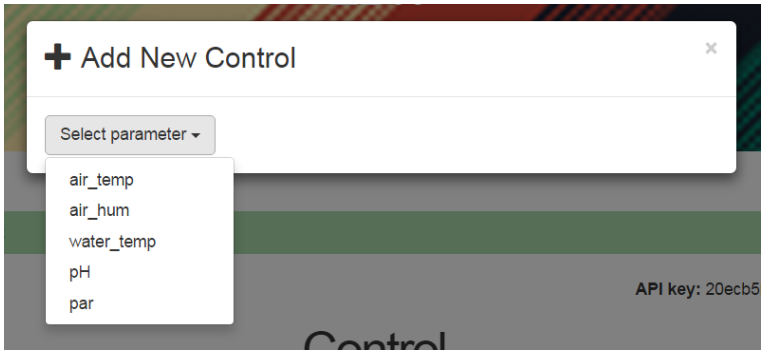
```
{
    "api_key": "115f64ee8ea246ac8251d8453798094bac6207e0",
    "typ": "connection",
    "message": "established over WiFi"
}
```

oblika odgovora

200 - aplication/json

```
{
    "status" : "OK, log saved"
}
```

Dodatno smo implementirali tudi možnost kontroliranja parametrov, za kar smo v bazi ustvarili model Control. Slednjega lahko povežemo z enim od obstoječih parametrov, ki se nahajajo v sistemu (Slika 4.4) in mu določimo minimalno oz. maksimalno vrednost, ob katerih se v primeru izbrane možnosti



Slika 4.4: Izbira parametra

proži alarm, ki preko spletne pošte obvesti lastnika sistema o dogodku.

Pošiljanje spletne pošte smo implementirali ob podpori t. i. ActionMailer-ja, s pomočjo katerega smo definirali vsebino pošte in pošiljanje povezali preko ponudnika Gmail.

```
AlarmMailer.alarm_email(@system.user, data).deliver_now
```

Primer 4.12: Pošiljanje opozorila ob preseženi vrednosti

4.2 Baza

Za podporo aplikaciji smo uporabili podatkovno bazo PostgreSQL. Rails ima zelo dobro razvit vmesnik za upravljanje z bazami. Da lahko nemoteno komunicira z izbrano, smo si pomagali s t. i. Ruby gem-om, ki se imenuje pg, zato smo ga vključili v datoteko Gemfile.:

```
gem 'pg',                '0.18.1'
```

Primer 4.13: Vključevanje Gem-a

Novo vključeni gem namestimo z ukazom v terminalu

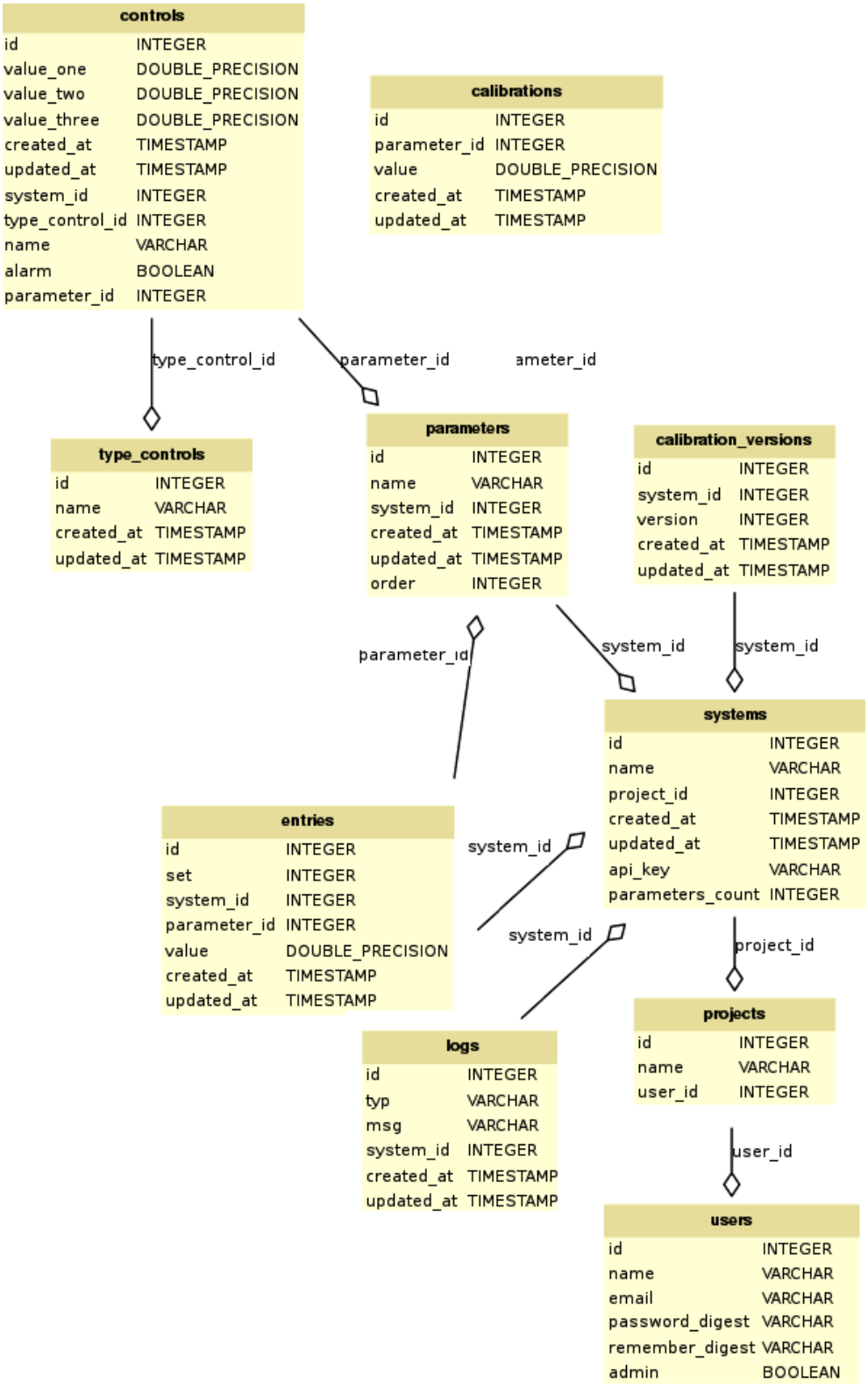
```
bundle install
```

Primer 4.14: Nameščanje gemov

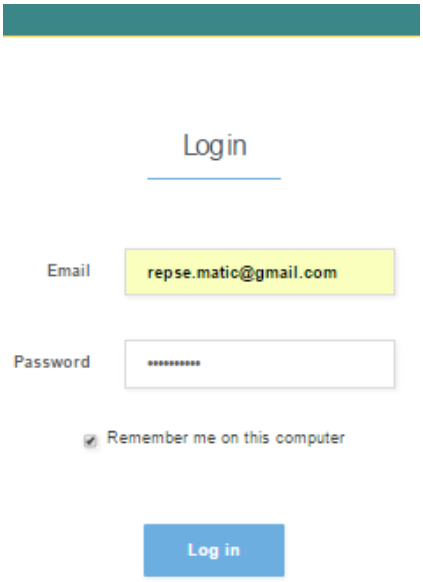
Ko je pg gem naložen, je za dostop do baze potrebno vnesti ustrezne podatke, ki so odvisni od sistema, kjer je baza nameščena in se nahajajo v datoteki database.yml. Shema ustvarjene baze je vidna na Sliki 4.5.

4.3 Primer uporabe

Po uspešno ustvarjenem uporabniškem računu in vpisu v aplikacijo (Slika 4.6), se znajdemo na strani, ki prikazuje dodane projekte(Slika 4.7). Če slednjih še nimamo jih lahko dodamo poljubno število s klikom na gumb Add project in podamo ime novega projekta. Ko se z miško približamo izbranemu projektu se pojavi napis VIEW PROJECT, na slednjega kliknemo in vstopimo v projekt. Ko se nahajamo v projektu imamo pred seboj seznam sistemov, katerih beležene vrednosti so predstavljene v tabelarični obliki (Slika 4.8). Če teh še nimamo, jih dodamo s klikom na gumb Add system in vnesemo ime novega sistema. Pred nami se pojavi nov prazen sistem, ki ne vsebuje nobenih parametrov (Slika 4.9). Da bi sistem dobil pomen, se s klikom na zobčasta kolesca, premaknemo na stran za urejanje sistema. Na slednji lahko sistemu dodamo parametre, ki jih po želji kalibriramo, dodamo kontrole, ki jih povežemo s prej ustvarjenimi parametri in pregledujemo zapise, ki se navezujejo na dani sistem (Slika 4.10). Po uspešno dodanih parametrih, lahko s poljubno implementacijo na mikrokrmilniku, ki uporablja API definiran višje v poglavju, v bazo podatkov začnemo zapisovati vrednosti. Slednje si lahko s klikom na graf hkrati ogledamo v tabelarični in grafični obliki (Slika 4.1). Poleg APIja za zapis prebranih vrednosti, lahko s poljubno implementacijo na MCUju uporabljamo tudi vse ostale višje omenjene funkcionalnosti.

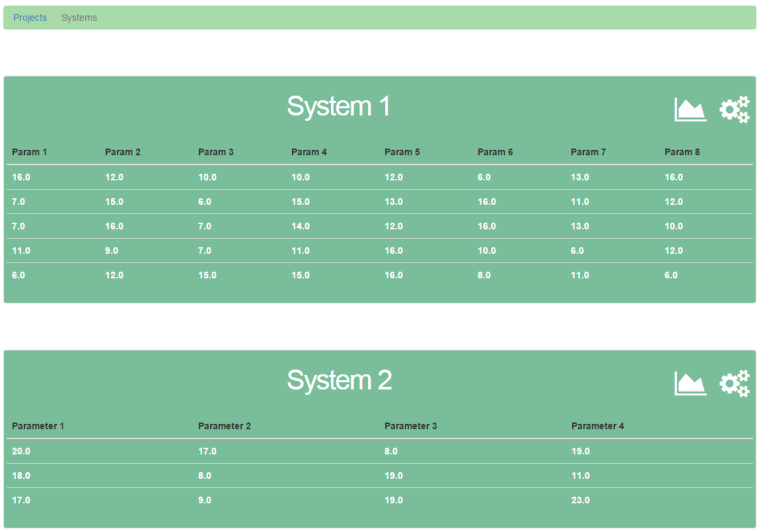


Slika 4.5: Shema baze

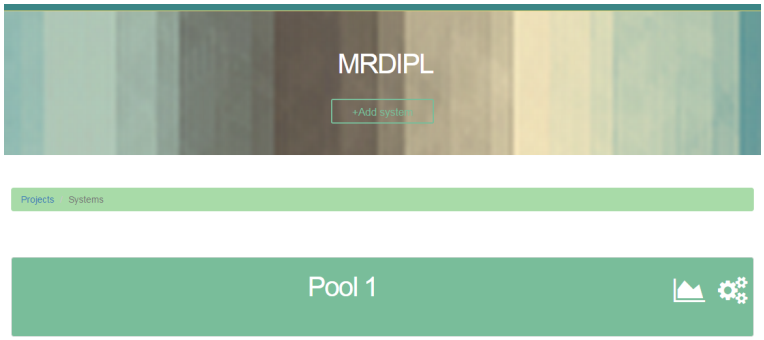


Slika 4.6: Vpis v aplikacijo

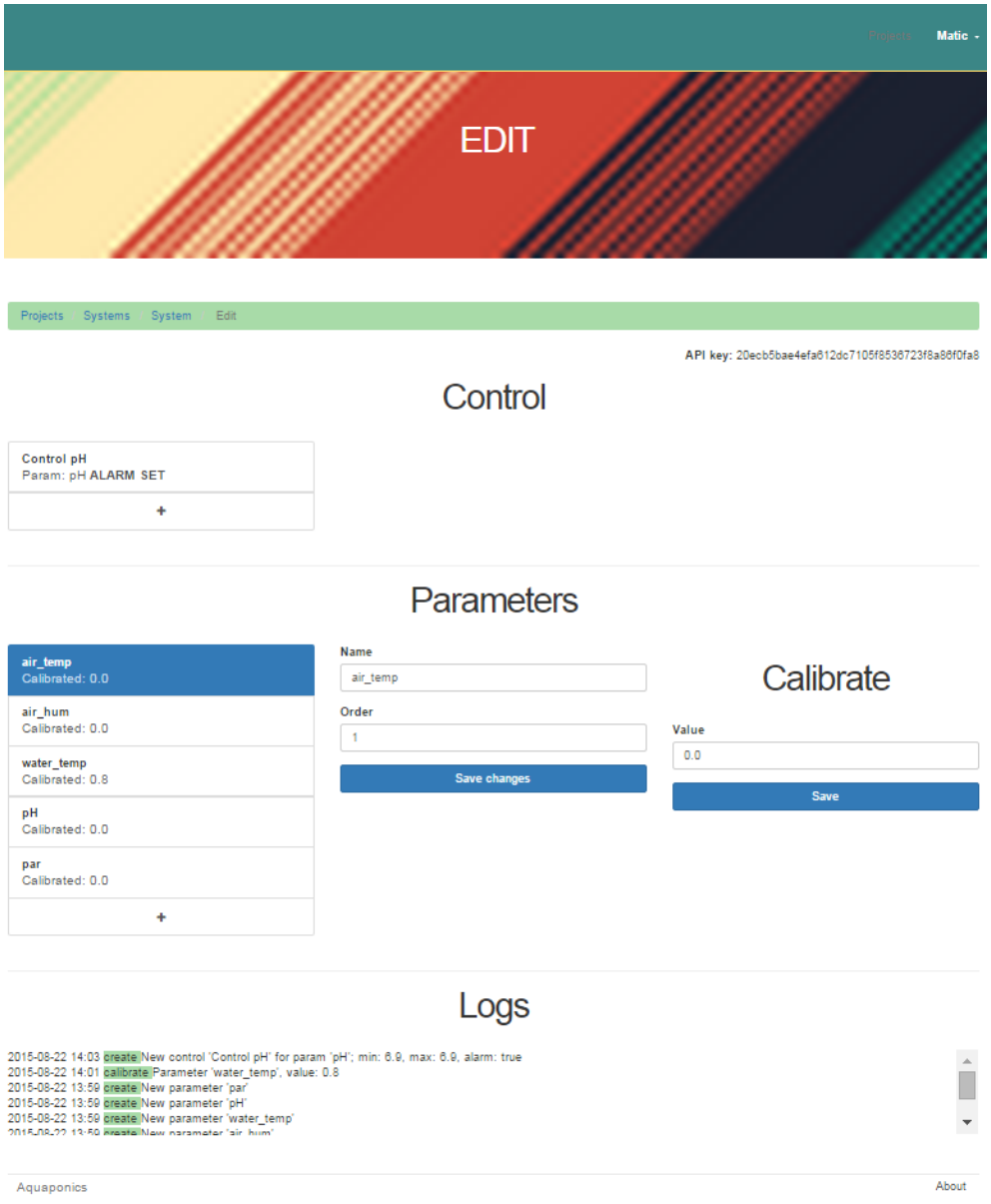
Slika 4.7: Seznam projektov



Slika 4.8: Seznam sistemov



Slika 4.9: Prazen sistem



Slika 4.10: Uredi sistem

Poglavje 5

Zaključek

V diplomskem delu smo opisali področje akvaponike in opredelili problematiko, s katero se ob tem srečujemo ter podali možno rešitev. Predstavili smo izbiro strojne opreme, s katero zajemamo podatke in nadziramo sistem ob pomoči mikrokrmilnika, kar smo dodatno podprli z izseki programske in psevdo kode, diagrami in omembo uporabljenih knjižnic. Prav tako smo natančno predstavili tiste elemente, ki smo jih razvili sami, tj. PAR senzor, pH vezje, avtomatski hranilnik ter na ta način dodatno znižali stroške končne rešitve. Naknadno smo za potrebe celotnega napajanja priredili ATX napajalnik, ki ga navadno najdemo v računalniških ohišjih. Največjo dodano vrednost ima pri rešitvi, kot posamezna komponenta, na dotik občutljiv zaslon; implementacija slednjega je bila časovno zahtevna, a kot komponenta nudi izjemno dobro uporabniško izkušnjo na mestu nadzora in je nepogrešljiv zaradi možnosti, ki jih ponuja.

V podporo strojni opremi smo razvili spletno aplikacijo, katere strežniški del smo napisali v jeziku Ruby, kot ogrodje pa smo uporabili Rails. Za podporo shranjevanja podatkov smo uporabili bazo PostgreSQL, ki je poleg tega, da omogoča vse, kar potrebujemo, podprta tudi s strani Heroku storitve, kamor smo postavili našo aplikacijo. Slednja zajema vse poglavitne funkcionalnosti, ki so potrebne v danem okolju in tako omogoča upravljanje s projekti in sistemi, grafičen in tabelaričen prikaz podatkov, kalibriranje pa-

rametrov, beleženje izvedenih akcij in obveščanje uporabnika ob doseženih mejnih vrednostih, kar pripomore k celotni varnosti sistema.

5.1 Nadaljnje delo

Kljub temu, da naš končni izdelek predstavlja zaključeno celoto, ki zadostuje za osnovni nadzor akvaponskega sistema, ostaja dovolj prostora za nadaljne razširitve:

- Na strani strojne opreme:
  - odpiranje oken s pomočjo linearnih aktuatorjev;
  - nadzorovan dotok vode preko magnetnih ventilov, s katero nadomeščamo izgubo;
  - nadzor zračnih črpalk, ki v bazene dodajajo raztopljen kisik;
  - vključevanje kamere, za vizualno spremljanje napredka rasti pridelka.
- Na strani aplikacije:
  - podpora za shranjevanje in prikaz slik;
  - vključevanje umetne inteligence za odločanje o izvedbi akcij, katerih rezultat je posredovan mikrokontrolerju.

Predstavljenih je zgolj nekaj idej, seznam je v resnici lahko še mnogo daljši.

Literatura

[1] Abu dhabi aquaponics system. [Online]. Dosegljivo: `http://www.jbauae.com/`. [Dostopano 4. 7. 2015].

[2] Arduino mega 2560. [Online]. Dosegljivo: `https://www.arduino.cc/en/Main/arduinoBoardMega2560/`. [Dostopano 4. 7. 2015].

[3] Stanford study: Half of fish consumed globally is now raised on farms. *States News Service*, 2009.

[4] A. Barth. Http state management mechanism. [Online]. Dosegljivo: `http://www.rfc-editor.org/rfc/rfc6265.txt`, 2011. [Dostopano 20. 7. 2015].

[5] Chitj Chauhan. *PostgreSQL Cookbook : Over 90 Hands-on Recipes to Effectively Manage, Administer, and Design Solutions Using PostreSQL.* Packt Publishing, 2015.

[6] Steve Diver. *Aquaponics-Integration of hydroponics with aquaculture.* Attra, 2000.

[7] David Flanagan and Yukihiro Matsumoto. *The ruby programming language.* "O'Reilly Media, Inc.", 2008.

[8] Michael Hartl. *Ruby on rails 3 tutorial : learn Rails by example / Michael Hartl.* Addison-Wesley professional Ruby series. Upper Saddle River, NJ : Addison-Wesley, c2011., 2011.

[9] Paul Horowitz and Winfield Hill. *The art of electronics.* Cambridge Univ. Press, 1989.

[10] Chris Kemp and Brad Gyger. *Professional Heroku Programming : An Architect's Guide.* Wrox, 2013.

[11] Marjetka Krese and Alenka Babšek. *Hidroponika.* Zbirka nasvetov: 21. Ljubljana : Kmečki glas, 1989 (Ljubljana : Učne delavnice), 1989.

[12] Brett A. Melbourne and Paul J. Daniel. A low-cost sensor for measuring spatiotemporal variation of light intensity on the streambed. *Journal of the North American Benthological Society*, (1):143, 2003.

[13] Odd Jostein Svendsli. Atmel's self-programming flash microcontrollers. *Systementwicklungsprojekt-Thomas Kittel*, 2013.

[14] Petr Vanysek. The glass ph electrode. *Electrochemical Society Interface*, page 19, 2004.

Dodatek A

Mikrokrmilnik

Ethernet

```
connect_client("POST", "/entries/save_multiple", data);
int connect_client(String method, String address, String data)
{
  if (client.connect(server, 3000))
  {
    client.println(method + " " + address + " HTTP/1.1");
    client.println("Host: 10.0.0.146");
    client.println("Accept: application/json");
    client.println("Connection: close");
    if (data.length() == 0)
    {
      client.println();
    }
    else
    {
      client.print("Content-Type: application/json\n");
      client.print("Content-Length: ");
      client.print(data.length());
      client.print("\n\n");
      client.print(data);
    }
    return 1;
  }
  else
  {
    return 0;
  }
}
```

Primer A.1: HTTP zahtevek preko ethernet

Wifi

```
wifi_post(data);

void wifi_post(String data)
{
  String cmd = "AT+CIPSTART=4,\"TCP\",\"";
  cmd += server_raw;
  cmd += "\",3000";
  sendData(cmd, 2000, true);
  String post = "POST /entries/save_multiple\n";
  post += "Host: 10.0.0.146\n";
  post += "Accept: application/json\n";
  post += "Connection: close\n";
  post += "Content-Type: application/json\n";
  post += "Content-Length: ";
  post += data.length();
  post += "\n\n";
  post += data;
  cmd = "AT+CIPSEND=" + String(post.length());
  sendData(cmd, 2000, true);
  sendData(post, 2000, true);
  sendData("AT+CIPCLOSE=4", 2000, true);
}

String sendData(String command, const int timeout, boolean debug)
{
    String response = "";
    ESP8266.println(command);
    long int time = millis();
    while( (time+timeout) > millis())
    {
      while(ESP8266.available())
      {
        char c = ESP8266.read();
        response+=c;
      }
    }
    return response;
}
```

Primer A.2: HTTP zahtevek preko WiFi

Kalibracija pH

```
int ph_high = 0;
int ph_low = 0;
void ph_calibration()
{
  screen_info("Put probe in pH 7 sol.", CENTER, MIDDLE, CLEAR);
  screen_info("Press continue and", CENTER, MIDDLE + 15, NO_CLEAR);
  screen_info("wait for 90 seconds.", CENTER, MIDDLE + 30, NO_CLEAR);
  draw_button(70, MIDDLE + 100, 170, MIDDLE + 135, "Continue", 90, MIDDLE
      + 112);
  while (!ph_continue){
    touch();
  }
  ph_continue = false;
  screen_info("Obtaining data.", CENTER, MIDDLE, CLEAR);
  delay(60000);
  for (int i = 0; i < 30; i++)
  {
    int ph = analogRead(10);
    ph_high = ph_high + ph;
    delay(1000);
  }
  ph_high = ph_high / 30;

  screen_info("Rinse probe with H20.", CENTER, MIDDLE, CLEAR);
  screen_info("When done press con.", CENTER, MIDDLE + 15, NO_CLEAR);
  draw_button(70, MIDDLE + 100, 170, MIDDLE + 135, "Continue", 90, MIDDLE
      + 112);
  while (!ph_continue) {
    touch();
  }
  ph_continue = false;

  screen_info("Put probe in pH 4 sol.", CENTER, MIDDLE, CLEAR);
  screen_info("Press continue and", CENTER, MIDDLE + 15, NO_CLEAR);
  screen_info("wait for 90 seconds.", CENTER, MIDDLE + 30, NO_CLEAR);
  draw_button(70, MIDDLE + 100, 170, MIDDLE + 135, "Continue", 90, MIDDLE
      + 112);
  while (!ph_continue) {
    touch();
  }
  ph_continue = false;
  screen_info("Obtaining data.", CENTER, MIDDLE, CLEAR);
  delay(60000);
  for (int i = 0; i < 30; i++)
  {
```

```
    int ph = analogRead(10);
    ph_low = ph_low + ph;
    delay(1000);
  }
  ph_low = ph_low / 30;

  int q = (ph_high - ph_low) / 3;
  ph_low = ph_high - (7 * q);
  ph_high = ph_high + (7 * q);
  EEPROM.updateInt(ADDRESS_PH_LOW, ph_low);
  EEPROM.updateInt(ADDRESS_PH_HIGH, ph_high);
  screen_info("Calibration done.", CENTER, MIDDLE, CLEAR);
  delay(2000);
  current_window = MAIN;
}
```

Primer A.3: Kalibracija pH

PAR

```
int read_par()
{
  int par = 0;
  long avg = 0;
  int i = 0;
  while (i < 100)
  {
    int r = analogRead(15);
    avg += r;
    i++;
    delay(5);
  }
  avg = avg / 100;
  if (avg > 500)
  {
    par = avg * 1.91;
  }
  else if (avg <= 500)
  {
    par = avg * 1.97;
  }
  return par;
}
```

Primer A.4: Branje senzorja in izračun PAR

Dodatek B

Strežnik

Project delec

```
<div id="project-<%= project.id %>" class="col-xs-6 col-md-4 projects">
  <div class="thumbnail orange-background">

    <b> <%= link_to project.name, project %> </b>
    <div>
      <b> <%= pluralize(project.systems.count, "system") %> </b>
    </div>
  <% if false %>
  <% if current_user?(project.user) %>
    <%= link_to "delete", project, method: :delete,
                data: { confirm: "You sure?" } %>
  <% end %>
  <% end %>

  </div>
</div>
```

Primer B.1: "Partial" predstava projekta

Obrazec

```
<%= form_for @project, remote: true, html: { class: "form-horizontal",
    style: "display:inline;" } do |f| %>
  <div class="modal-body">
    <div id="error_explanation" class="bg-danger text-danger">
</div>
```

```
    <div class="control-group">
      <%= f.label :name, class:"control-label" %>
      <div class="controls">
        <%= f.text_field :name %>
      </div>
    </div>

  </div>
  <div class="modal-footer">
    <%= f.submit class: "btn btn-primary" %>
    <%= link_to "Cancel", "#", class: "btn", data: {dismiss: "modal"} %>
  </div>
<% end %>
```

Primer B.2: Primer obrazca

Highcharts

```
$(function () {
    var parameters = <%= parameters.to_json.html_safe %>;
    var data = <%= data.to_json.html_safe %>;
    var plot = {};
    $.each(data, function(key, value) {
        $.each(value, function(k, v) {
            if (k in plot) {
                plot[k].push([parseInt(key), v]);
            } else {
                plot[k] = [];
                plot[k].push([parseInt(key), v]);
            }
        });
    });
    var plot_data = [];
    $.each(plot, function(k, v) {
        console.log(k, v);
        for (var i=0; i<parameters.length; i++)
          if (parameters[i].id == k)
            var name = parameters[i].name;
        plot_data.push({name: name, data: v});
    });
    $('#cont').highcharts({
        credits: {
            enabled: false
        },
        title: {
```

```
            text: 'Parameters',
            x: -20
        },
        subtitle: {
            x: -20
        },
        xAxis: {
            title: {
                text: 'Set'
            },
        },
        yAxis: {
            plotLines: [{
                value: 0,
                width: 1,
                color: '#808080'
            }]
        },
        legend: {
            layout: 'vertical',
            align: 'right',
            verticalAlign: 'middle',
            borderWidth: 0
        },
        series: plot_data
    });
});
```

Primer B.3: Uporaba knjižnice highcharts

API ključ

```
class System < ActiveRecord::Base
  after_create :generate_api_key
  belongs_to :project
  validates :project_id, presence: true

  def generate_api_key
    to_digest = self.project.user.email + self.created_at.to_s
    self.api_key = Digest::SHA1.hexdigest to_digest
    self.save
  end
end
```

Primer B.4: Generiranje API ključa