Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matej Repinc

Vremenska postaja s spletnim strežnikom

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: prof. dr. Patricio Bulić

Ljubljana 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00421/2013
Datum: 05.04.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATEJ REPINC**

Naslov: **VREMENSKA POSTAJA S SPLETNIM STREŽNIKOM**
WEATHER STATION WITH A WEB SERVER

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Izdelajte avtomatsko vremensko postajo s spletnim strežnikom. Za izdelavo merilnega modula uporabite razvojni sistem Arduino z mikrokrmilnikom ATmega328 ter tipala za temperaturo, vlažnost in tlak. Strežniški del postaje zgradite na odprtokodnem spletnem strežniku Apache, ki ga poganja računalnik z operacijskim sistemom Linux. Tako izdelana vremenska postaja naj omogoča časovno beleženje izmerjenih parametrov ter dostop do njih preko spleta v realnem času. Vso programsko opremo objavite pod odprtokodno licenco GNU LGPL in jo naredite javno dostopno na repozitoriju GitHub.

Mentor:
izr. prof. dr.Patricio Bulić

Dekan:
prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Matej Repinc, z vpisno številko **63090307**, sem avtor diplomskega dela z naslovom:

Vremenska postaja s spletnim strežnikom

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Patricia Bulića,
- 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 v zbirki "Dela FRI".

V Ljubljani, dne 5. septembra 2013 Podpis avtorja:

Hvala vsem, ki ste mi v času študija stali ob strani in me podpirali v dobrem in slabem. Hvala tudi prof. Buliću pri pripravi diplomske naloge.

Kazalo

Povzetek

V diplomski nalogi vam bomo predstavili izdelavo ter funkcionalnost poceni vremenske postaje, ki temelji na prototipni platformi Arduino. Vremenska postaja spremlja trenutne podatke o temperaturi ter vlažnosti zraka, zračnemu pritisku ter izračuna vrednost rosišča. Postaja ima tudi svoj preprost strežnik HTTP, prek katerega posreduje trenutne podatke v dveh različnih formatih: kodirani podatki JSON ter spletna stran HTML. Prav tako zna vremenska postaja podatke pošiljati na strežnik za zbiranje podatkov. Drugi del naloge je bila izdelava spletne strani za prikaz trenutnih podatkov in statističnih grafov. Spletna stran je narejena s kombinacijo tehnologij HTML, CSS, Javascript ter PHP. Vse komponente naloge povezuje programski vmesnik (angl. API - Application Programming Interface), ki služi vnašanju in pridobivanju podatkov iz podatkovne baze. Vsi sklopi programske kode so zasnovani tako, da se jih kar najlažje da zamenjati z drugimi komponentami.

Ključne besede: *vremenska postaja, programski vmesnik, spletna stran, merjenje vremenskih podatkov, arduino.*

Abstract

In this diploma thesis we present the process of making a cheap weather station using Arduino prototyping platform and its functionality. The weather station monitors current temperature, humidity of air and air pressure. The station has its own simple HTTP server that is used to relay current data in two different formats: JSON encoded data and simple HTML website. The weather station can also send data to a pre-defined server used for data collection. We implemented a web site where data and graphs can be viewed in a user friendly way. The website was created using a combination of HTML, CSS, Javascript and PHP. We also implemented a programming interface (API) used for collecting data and querying database, which connects all of the pieces together.

Keywords: *weather station, application interface, web site, weather data measurement, arduino.*

Poglavje 1

Uvod

Na trgu obstaja veliko digitalnih avtomatskih vremenskih postaj. Za te je značilno, da nimajo povezave z računalnikom, tiste, ki pa jo imajo, pa niso prav poceni, za dodajanje različnih funkcionalnosti pa moramo pogosto dokupiti določene programske ter strojne module. V diplomski nalogi smo izdelali vremensko postajo na 'odprtokodni' način. Vse uporabljene komponente lahko dobimo iz različnih spletnih trgovin, vsa koda je odprtokodna in na voljo v repozitoriju GitHub (`https://github.com/jernejovc/arduinows`), uporabljeni strežniki pa so dostopni brezplačno in so tudi odprtokodni. Torej je bil naš cilj že od začetka čim večja modularnost (in posledično tudi zamenljivost) vseh uporabljenih komponent. Koda, ki teče na platformi Arduino, je izdelana tako, da je del kode za branje senzorjev ločen od dela, ki pošilja podatke na zunanji strežnik, tako da lahko uporabimo praktično vsak senzor, če le znamo prebrati njegovo vrednost z mikrokrmilnikom Arduino. Za prikaz podatkov v človeku bolj prijazni obliki je uporabniku na voljo spletna stran, ki prikazuje podatke iz vremenske postaje v obliki grafov. Na spletni strani lahko prav tako pregledujemo zgodovinske podatke (npr. gibanje temperature v določenem mesecu). Vse programske komponente povezuje programski vmesnik, prek katerega pošiljamo ter pridobivamo podatke.

Poglavje 2

Odprtokodne licence

Ker smo vremensko postajo ter pripadajočo programsko opremo izdelali v odprtokodnem duhu, smo se odločili da besedilo diplomske naloge in kodo vremenske postaje ter spletne strani objavimo pod odprtokodnimi licencami. Pri izbiri licence smo gledali najprej na pogoje licence, faktor izbire pa je bilo tudi odobravanje različnih odprtokodnih ter legalnih društev. Odločili smo se za uporabo dveh licenc, Creative Commons ter GNU Lesser General Public License različice 3.

2.1 Creative Commons licenca

Besedilo diplomske naloge je na voljo pod Creative Commons CC BY-SA licenco. Creative Commons je licenca, ki je največkrat uporabljena za licenciranje besedila, slik, glasbe ... Izbira te licence pomeni, da lahko besedilo kdorkoli uporablja oziroma kopira, mora pa navesti avtorja. V primeru, da nekdo delo spremeni ter ga javno objavi, mora poskrbeti da, to delo objavi pod istimi pogoji, torej pod licenco CC BY-SA.
Creative Commons licenca je sicer sestavljena iz štirih pogojev [1], ki se jih lahko poljubno kombinira:

- BY - Atribucija: Uporabnik lahko delo uporablja v javne namene, mora pa poskrbeti da, navede originalnega avtorja ter licenco;

- SA - Deljenje pod istimi pogoji: Uporabnik dela lahko deli derivate originalnega dela pod istimi pogoji, kot jih določa licenca (npr. če je delo licencirano s CC BY-SA, morajo biti vsi derivati dela licencirani z licenco CC BY-SA);

- NC - Nekomercialno: Uporabnik dela ne sme uporabiti v komercialne namene;

- ND - Brez derivatov dela: Uporabnik delo uporablja, ne sme pa delati derivatov dela.

Vseh kombinacij licence je torej 16, od katerih je 11 veljavnih, 5 pa ne (4 kombinacije z ND ter SA, ki se logično izključujeta, ter prazna kombinacija).

2.2 GNU Lesser General Public License 3

Kodo vremenske postaje, vmesnika API ter spletne strani smo objavili pod GNU Lesser General Public License različice 3 (GNU LGPLv3). GNU licence se največkrat uporabljajo za licenciranje izvorne kode programov. GNU LGPLv3 je nekoliko blažja oblika licence GNU General Public License različice 3 (GNU GPLv3). Kodo, objavljeno pod licenco GNU GPLv3, lahko drugi uporabniki uporabljo z drugimi kosi programske opreme, ta programska oprema pa mora biti prav tako odprtokodna ter licencirana z isto oziroma z legalno skladno licenco [2]. GNU LGPLv3 pa določa, da se lahko koda, licencirana s to licenco, uporablja tudi skupaj z lastniško (zaprtokodno) programsko opremo [3]. V tem primeru mora biti jasno navedeno, da se uporablja odprtokodna programska oprema [3].
Razliko lahko lepo predstavimo na primeru programskih knjižnic (`.so` knjižnice na platformi Linux oziroma `.dll` knjižnice na platformi Microsoft Windows). Vzemimo primer, kjer naredimo programsko knjižnico ter jo licenciramo z licenco GNU GPLv3. Potem mora biti mora vsa programska oprema, ki želi uporabljati to knjižnico, licencirana pod pogoji GNU GPLv3 oziroma podobne odprtokodne licence. Če pa knjižnico licenciramo z licenco GNU LGPLv3, lahko knjižnico uporablja vsak, mora pa jasno povedati, da je uporabljena tudi programska oprema z licenco GNU LGPLv3.

V našem primeru to pomeni, da lahko katerikoli del programske opreme uporabljajo tudi zaprtokodni programi. Primer tega je npr., da lahko katerakoli tretja oseba naredi zaprtokodno programsko opremo, ki uporablja naš programski vmesnik za pridobivanje oziroma pošiljanje vremenskih podatkov na strežnik.

2.2.1 Alternative GNU LGPLv3

Obstaja seveda veliko alternativnih odprtokodnih licenc, ki so nastale za potrebe različnih programskih paketov ali organizacij. Licence se med seboj razlikujejo predvsem v tem, koliko "omejujejo" uporabnika programske opreme. Razlika je večinoma v tem, kako se lahko dela, licencirana s posamezno licenco, uporabljajo v kombinaciji z deli, ki so zaprtokodni, načinu distribucije, ali lahko dodajamo svoje pogoje k določeni licenci (to npr. omogoča Apache License 2.0 [4] [5]). Nekatere licence, npr. MIT License, ne omejujejo uporabe programske opreme oziroma derivativov na kakršenkoli način [6] [7]. Večina oziroma praviloma vse licence prav tako določajo, da se mora ob distribuciji programske opreme podati tudi določena datoteka, v kateri je zapisana licenca. Primerjavo nekaterih najbolj uporabljenih odprtokodnih licenc prikazuje tabela 2.1.

Tabela 2.1: Primerjava odprtokodnih licenc [5] [8] [9] [10] [7] [11]

Legenda

L	Lahko
M	Mora
N	Ne sme / Ne odgovarja
/	Ne določa
*	Omejeno
†	Uporabnik lahko brez ponovnega prevajanja programa uporabi drugo knjižnico ali različico knjižnice

Licenca	Spremembe	Komercialna uporaba	Distribucija	Ponovno licenciranje	Garancija	Uporaba zaščitnega znaka	Sodno odgovarjanje	Dodajanje obvestila o avtorskih pravicah	Dodajanje licence	Dodajanje originalne kode	Prikaz svoje kode	Navajanje sprememb	Nestatično povezovanje†
Apache	L	L	L	L	L	N	N	M	M	/	/	M	/
BSD 3	L	L	L	L	L	N	N	M	/	/	/	/	/
GPLv3	L	L*	L	N	L	/	N	/	/	M	M	M	/
LGPLv3	L	L*	L	L	L	/	N	/	/	M	/	/	M
MIT	L	L	L	L	/	/	N	M	/	/	/	/	/
Mozilla Public License	L	L	L	L	/	N	N	M	M	/	M	/	/

Poglavje 3

Uporabljene komponente

3.1 Strojne komponente

Pri izbiri strojnih komponent se je vse vrtelo okoli izbranega mikrokrmilnika, to je prototipna platforma Arduino. Naš glavni cilj pri izbiri komponent je bil čim večji izkoristek mikrokrmilnika in poraba čim manj programskega prostora. Platforma Arduino omogoča tako priklop digitalnih kot analognih senzorjev, za potrebe našega projekta smo se odločili za izbiro digitalnih senzorjev, prednost pa smo dali senzorjem, ki uporabljajo vodilo I^2C (two-wire interface). To vodilo nam omogoča priklop več naprav na eno vodilo, s tem prihranimo kar nekaj prostora, saj nam ni treba vsakega senzorja priključiti na svoj digitalni pin na mikrokrmilniku, ampak vse povežemo na eno vodilo.

3.1.1 Arduino

Arduino je mikrokrmilnik, ki smo ga izbrali za potrebe diplomske naloge. Model, ki je bil izbran, se imenuje Arduino Ethernet z naslednjimi karakteristikami:

- Mikrokrmilnik ATmega328 [12];
- 32 KB pomnilnika flash, uporabljenega za shranjevanje programa [12];
- 2 KB pomnilnika SRAM, uporabljenega za run-time spremenljivke [12];

- 1 KB pomnilnika EEPROM, uporabljenega za shranjevanje podatkov, ki se tekom izvajanja ne spreminja [12];
- 16 MHz hitrost ure procesorja [12];
- Modul Ethernet, s katerim se povezujemo v omrežje [12];
- Modul PoE [12], ki zagotavlja napajanje prek kabla Ethernet [13].

Eden ključnih dejavnikov izbire tega modela mikrokrmilnika je bil prav gotovo modul PoE (ang. Power over Ethernet). Ta nam omogoča, da mikrokrmilnik napajamo samo preko kabla Ethernet [13]. Prednost tega pristopa je ta, da pri inštalaciji vremenske postaje potrebujemo samo en kabel, ki služi tako za komunikacijo, kot tudi za napajanje celotne vremenske postaje.
Za komunikacijo s senzorji smo se odločili za uporabo protokola oziroma vodila I^2C z uporabo knjižnice `Wire` [14] oziroma za uporabo drugih digitalnih vodil ali senzorjev.

I^2C

I^2C je sinhrono vodilo, ki nam omogoča priklop relativno počasnih naprav na hitre mikrokrmilnike.
I^2C za komunikacijo uporablja 2 signala:

1. SCK (Serial Clock), ki nosi frekvenco ure, in
2. SDA (Serial Data), preko katerega se prenašajo podatki.

Kot smo omenili že prej, je vodilo sinhrono, kar pomeni, da se podatki in ura prenašajo hkrati, veljavnost podatkov pa se določa s pozitivno ali negativno fronto ure. Iz tega sledi, da sta pri sinhronem vodilu periodi ure ter podatkov usklajeni.

Na vodilo sta lahko priključeni dve vrsti naprav:

1. Gospodar (angl. master): Naprava, ki generira urin signal SCK in ga pošilja na žico ter začenja komunikacijo z napravami suženj;

2. Suženj (angl. slave): Naprava, ki prejema urin signal SCK in se odzove, ko ga naprava gospodar naslovi.

Vodilo je tipa multi-master, kar pomeni da je na vodilo lahko priklopljeno več naprav, ki so t. i. "master", in lahko od drugih naprav zahtevajo podatke ali v naprave posredujejo podatke. Na žici je lahko na enkrat aktivna samo ena "master" naprava.
Vsaka naprava ima tudi svoj naslov, prek katerega je enolično identificirana na vodilu. Specifikacija vodila določa, da mora naprava, ki je naslovljena z omenjenim naslovom, vodilo ustrezno spremljati ter sprejeti oziroma oddati podatke.

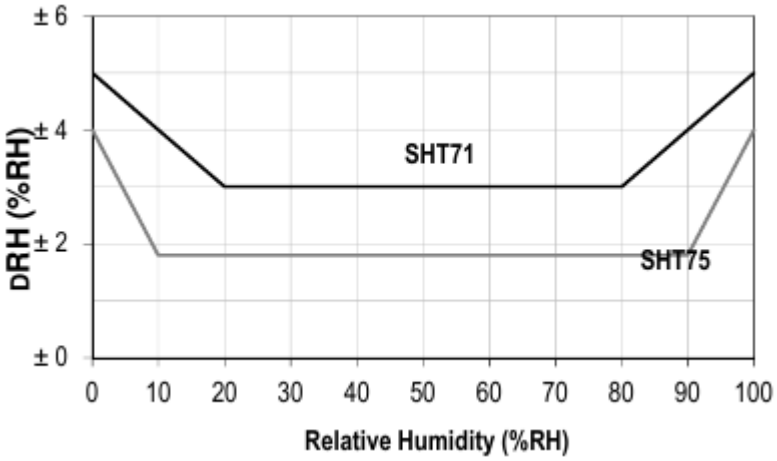
3.1.2 Senzor temperature in vlažnosti

Za senzor temperature ter vlažnosti smo si izbrali SHT75 proizvajalca Sensirion [15]. SHT75 je digitalni senzor, ki pa za razliko od senzorja BMP085 ne uporablja vodila I^2C, ampak moramo za komunikacijo poskrbeti sami. Senzor SHT75 je bil izbran zaradi natančnega merjenja ter dokaj ugodne cene. Karakteristike senzorja SHT75 prikazuje tabela 3.1.

Tabela 3.1: Karakteristike senzorja SHT75 [16].

Nazivna napetost [V]	3.3
Vodilo	Digitalno, dve žici: Data in Clock
Obseg merjenja vlažnosti [%RH]	$0 - 100$
Resolucija merjenja vlažnosti [%RH]	0.4 (pri 8 bitih), 0.05 (pri 12 bitih)
Natančnosti merjenja vlažnosti [%RH]	Tipično ± 1.8, maksimalno pri 25 °C: glej sliko 3.1
Resolucija merjenja temperature [°C]	0.04 (pri 12 bitih), 0.01 (pri 14 bitih)
Natančnost merjenja temperature [°C]	Tipično ± 0.4, maksimalno: glej sliko 3.2
Obseg merjenja temperature [°C]	$-40 - 123.8$

Slika 3.1: Maksimalna toleranca relativne vlažnosti pri 25 °C za senzor SHT75 [16]



Kot nam prikazuje tabela 3.1, nam senzor omogoča dokaj natančno merjenje. Temperaturo lahko merimo z resolucijo 0.05 °C (pri uporabi 12 bitov za merjenje) oziroma 0.01 °C (pri uporabi 14 bitov za merjenje), vlažnost pa z resolucijo 0.4% (pri uporabi 8 bitov za merjenje) oziroma 0.05 % (pri uporabi 12 bitov za merjenje).

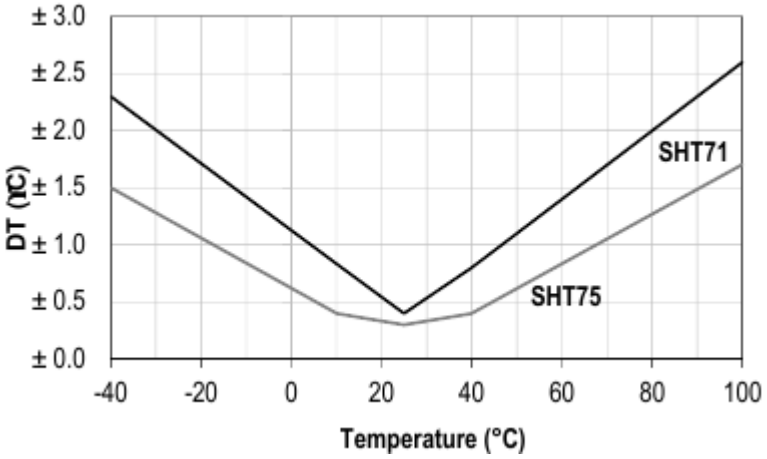
Ob nakupu oziroma izbiri senzorja smo spregledali dejstvo, da senzor ne uporablja digitalnega vodila I^2C, ampak digitalno dvo-žično povezavo. Tako ga nismo priklopili na vodilo I^2C, ampak na dva digitalna pina na mikrokrmilniku Arduino.

3.1.3 Senzor tlaka

Za potrebe diplomske naloge smo uporabili senzor tlaka BMP085 proizvajalca Bosch Sensortec.

Kot lahko razberemo iz tabele 3.2, nam senzor omogoča dokaj natančno merjenje pritiska v "normalnem" temperaturnem območju (za amaterske uporabnike vremenske postaje).
Senzor BMP085 ima prav tako vgrajen temperaturni senzor, s katerim določimo temperaturo in si s tem podatkom pomagamo pri izračunu referenčnega tlaka na višini $0m$. V našem primeru merjenja temperature s senzorja BMP085 nismo uporabljali, saj smo imeli na voljo natančnejši senzor SHT75.

Slika 3.2: Maksimalna toleranca temperature za senzor SHT75 [16]



3.2 Programske komponente

Kot smo omenili že v uvodu, smo izbrali lahko dostopne programske komponente. Vse uporabljene komponente programske opreme so odprtokodne. Naša primarna platforma sestoji iz treh glavnih komponent:

1. Spletni strežnik Apache;
2. Podatkovna baza MySQL;
3. Skriptni jezik PHP.

Te komponente smo izbrali zaradi več razlogov, glavni med njimi pa je bila odprtokodnost in prenosljivost med operacijskimi sistemi in platformami. V našem testnem sistemu smo sicer uporabljali distribucijo Linux CentOS[1], vse komponente pa bi se počutile prav tako domače tudi na kakšni drugi distribuciji Linux ali platformi Windows. Naš testni sistem je bil sestavljen iz dveh navideznih strežnikov, na enem je tekel spletni strežnik, na katerem sta bila postavljena spletna stran in programski vmesnik, na drugem pa podatkovna baza. Brez težav bi lahko vse komponente poganjali na enem ali še več navideznih ali fizičnih računalnikov ali strežnikov.

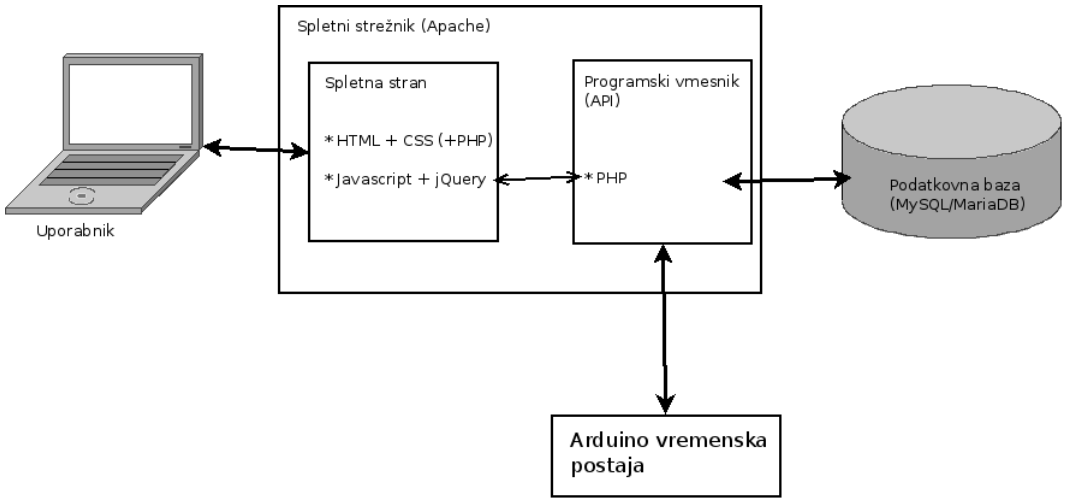
[1] `http://centos.org`

Tabela 3.2: Karakteristike senzorja BMP085 [17]

Nazivna napetost [V]	3.3
Vodilo	I^2C
Temperatura delovanja [°C]	$-40 - 85$
Temperatura delovanja s polno natančnostjo [°C]	$0 - 65$
Obseg merjenja pritiska [hPa]	$300 - 1100$
Natančnost merjenja pritiska ($700 -$ 1100 hPa, $0 - 65$ °C) [hPa]	± 2.5, tipično ± 1
Natančnost merjenja pritiska ($300 -$ 700 hPa, $0 - 65$ °C) [hPa]	± 3.0, tipično ± 1
Natančnost merjenja pritiska ($300 -$ 1100 hPa, $-20 - 0$ °C) [hPa]	± 4.0, tipično ± 1.5
Natančnost merjenja temperature (pri 25 °C) [°C]	± 1.5, tipično ± 0.5
Natančnost merjenja temperature (pri $0 - 65$ °C) [°C]	± 2.0, tipično ± 1.0

Slika 3.3 nam prikazuje odnos med strežniki, vremensko postajo ter uporabniki sistema. Na sliki so razvidne tudi komunikacijske poti, ki prikazujejo odnose med različnimi komponentami programske opreme, razvidno pa je tudi, katere tehnologije smo uporabili pri izdelavi različnih delov programske opreme. Tako na primer tudi vidimo, da uporabnik ali vremenska postaja nikoli ne uporablja direktne povezave s podatkovno bazo, ampak se za komunikacijo s podatkovno bazo uprablja programski vmesnik API. Prav tako spletna stran uporablja programski vmesnik za pridobivanje podatkov preko knjižnice jQuery oziroma preko asinhronskega klica "AJAX" (angl. Asynchronous Javascript And XML).

Slika 3.3: Arhitektura strežnikov, vremenske postaje ter uporabnika



3.2.1 Razvojno okolje

Pri razvoju programske opreme smo si pomagali z moderno programsko opremo, ki nam to delo olajša. Za razvoj velikih programskih rešitev je navadno značilno, da se uporablja integrirano programsko okolje ter programska oprema za nadzor različic. Pri razvoju spletne strani ter programskega vmesnika smo uporabili naslednja orodja:

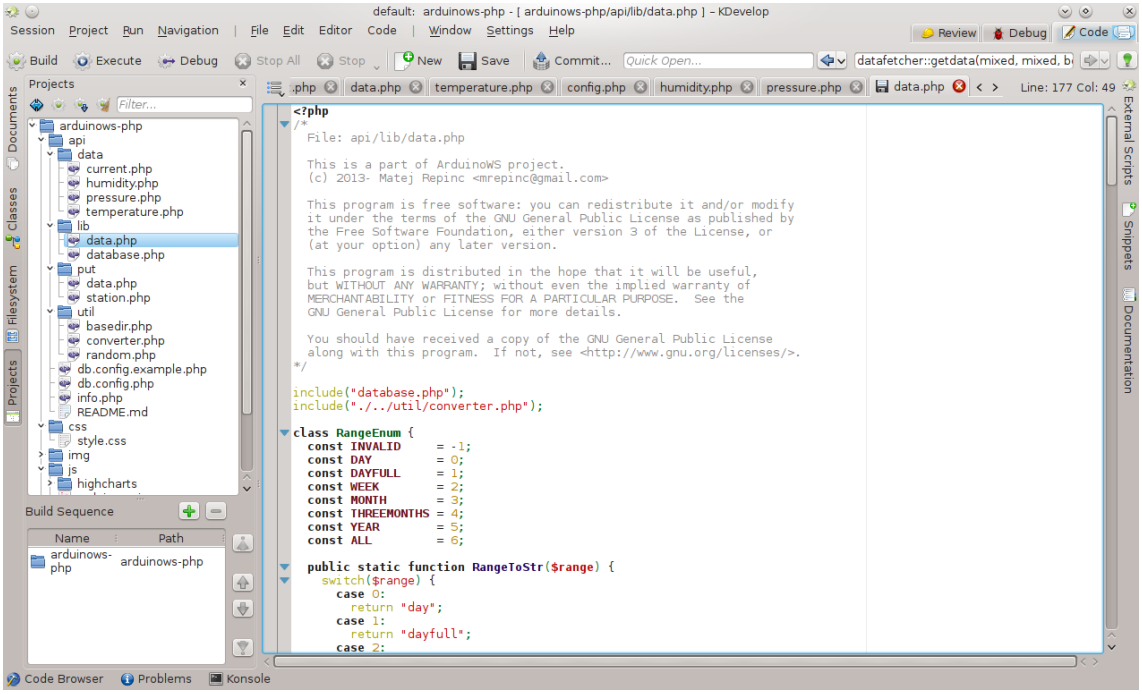
- KDevelop IDE (integrirano razvojno okolje, angl. Integrated Development Enviornment) [18]: programsko okolje, ki nam je olajšalo delo pri programiranju v programskem jeziku PHP. Orodje nam omogoča samodejno dopolnjevanje spremenljivk, razredov, funkcij ... kar pripomore k učinkovitejšemu delu. KDevelop prav tako omogoča delo s projekti, uvoz iz različnih virov, je hiter ter porabi sorazmerno malo sistemskih sredstev. Slika 3.4 nam prikazuje izgled okna urejevalnika v razvojnem pogledu ter pogled projekta.

- Git [19]: programska oprema za nadzor različic. Pri razvoju programske opreme se večkrat srečamo s problemom spremljanja razvoja programske opreme ter spremljanjem sprememb med razvojem. Git je majhen ter hiter

[20] sistem za nadzor različic, ki je uporabljen pri vseh vrstah projektov. Za razvoj naše programske opreme smo ga izbrali prav zaradi hitrosti ter majhne porabe sistemskih sredstev. Faktor izbire sistema Git pred drugimi podobnimi orodji je bilo tudi gostovanje kode naše programske opreme na spletni strani GitHub [21]. Sistem Git nam prav tako omogoča izdelavo oziroma pregled nad različicami (angl. version) programske opreme s pomočjo oznak (angl. tag).

- GitHub [21]: GitHub je spletna stran, ki nam omogoča gostovanje programske kode. Za dostop do kode lahko uporabljamo sistem Git ali spletni vmesnik. Spletna stran GitHub nam omogoča tako brezplačno gostovanje (pri čemer je koda javno vidna), kot tudi plačljivo gostovanje (v tem primeru je koda lahko tudi zasebna).

Slika 3.4: Zaslonski posnetek razvojnega okolja KDevelop



3.2.2 Spletni strežnik Apache

Prva različica spletnega strežnika Apache je bila izdana leta 1995 [22], od takrat pa je eden izmed najbolj priljubljenih spletnih strežnikov. Za uporabo spletnega strežnika Apache smo se odločili zaradi enostavne namestitve (v večini Linux distribucij je na voljo za preprosto namestitev prek upravljalca paketov) in majhne porabe sistemskih sredstev. Spletni strežnik Apache smo morali nastaviti tako, da zna procesirati datoteke PHP, na našem sistemu CentOS to pomeni namestitev programskega paketka `php` z upravljalcem paketkov. V bolj naprednem sistemu bi se lahko poslužili bolj napredne konfiguracije, na primer varnega dostopa do spletne strani in programskega vmesnika z uporabo protokola HTTPS (HTTP z uporabo SSL), a smo se odločili, da za trenutne potrebe takšnega dostopa ne podpremo.

3.2.3 Programski jezik PHP

Kot programski jezik za del spletne strani in celoten programski vmesnik API smo si izbrali PHP. PHP je objektno-orientiran, interpretiran odprtokodni programski jezik, ki ga razvija skupina "The PHP Group" [23]. Naša izbira je bila narejena predvsem zaradi predhodnega poznavanja programskega jezika. PHP je prav tako eden izmed najbolj uporabljenih strežniških skriptnih jezikov (uporabljen za izdelavo 80 % spletnih strani [24]), nekateri kazalci kažejo na to, da poganja več kot 244 milijonov spletnih strani in je nameščen na 2,1 milijonu naslovov IP, na katerih streže strežnik HTTP (Januar 2013 [25]). Prav tako je jezik PHP v zreli stopnji razvoja, to pomeni da je stabilen in da se aktivno razvija. Za razvoj programskega vmesnika s programskim jezikom PHP smo se odločili predvsem zaradi dobre podpore povezovanja in pridobivanja podatkov iz podatkovne baze MySQL. V primeru kasnejše podpore povezave na druge podatkovne baze, pa bi to tudi brez težav implementirali, saj ima jezik PHP že vgrajeno podporo tudi za druge sisteme za upravljanje s podatkovnimi bazami (angl. DBMS, DataBase Management System), na primer PostgreSQL [26] ali povezave preko vmesnika ODBC [27].

3.2.4 HTML + CSS + Javascript + jQuery

Pri izdelavi same spletne strani smo uporabili trenutno najbolj popularne komponente modernih spletnih strani. Za osnovo smo naredili statično stran HTML, ki smo jo oblikovali s pomočjo stilnih predlog CSS (angl. Cascading Style Sheets). Spletna stran je sicer ustvarjena s pomočjo programskega jezika PHP, ki doda še prilagoditve spletne strani, na primer ime vremenske postaje in njeno lokacijo oziroma opis. Za pridobivanje podatkov preko vmesnika API smo uporabili asinhronsko uporabo metode AJAX z uporabo knjižnice jQuery, ki nam glede na uporabo metode AJAX prek standardnih Javascript funkcij poenostavi rokovanje s podatki. Na spletni strani so uporabljeni tudi grafi za prikaz podatkov v vizualni obliki, za njihovo izdelavo smo uporabili knjižnico Highcharts [28].

3.2.5 Podatkovna baza MySQL/MariaDB

Glavni medij hrambe vseh podatkov je bil tako zaradi praktičnosti kot tudi performanc podatkovna baza. Za podatkovno bazo MySQL/MariaDB smo se odločili predvsem zaradi dobre "naveze" s programskim jezikom PHP, pa tudi zaradi dostopnosti programskega paketa. V zadnjem času se sicer vse bolj uveljavlja podatkovna baza MariaDB, ki je odprtokodna veja podatkovne baze MySQL, ki je nastala kmalu po tem, ko je podjetje Sun kupilo MySQL. Podatkovna baza MariaDB je deležna več popravkov in novih funkcionalnosti, saj razvoj poteka v čistem odprtokodnem duhu brez pristranosti podjetij. V teoriji bi morala funkcionalnost vremenske postaje ostati ista, ne glede na to ali uporabljamo MariaDB ali MySQL.

Poglavje 4

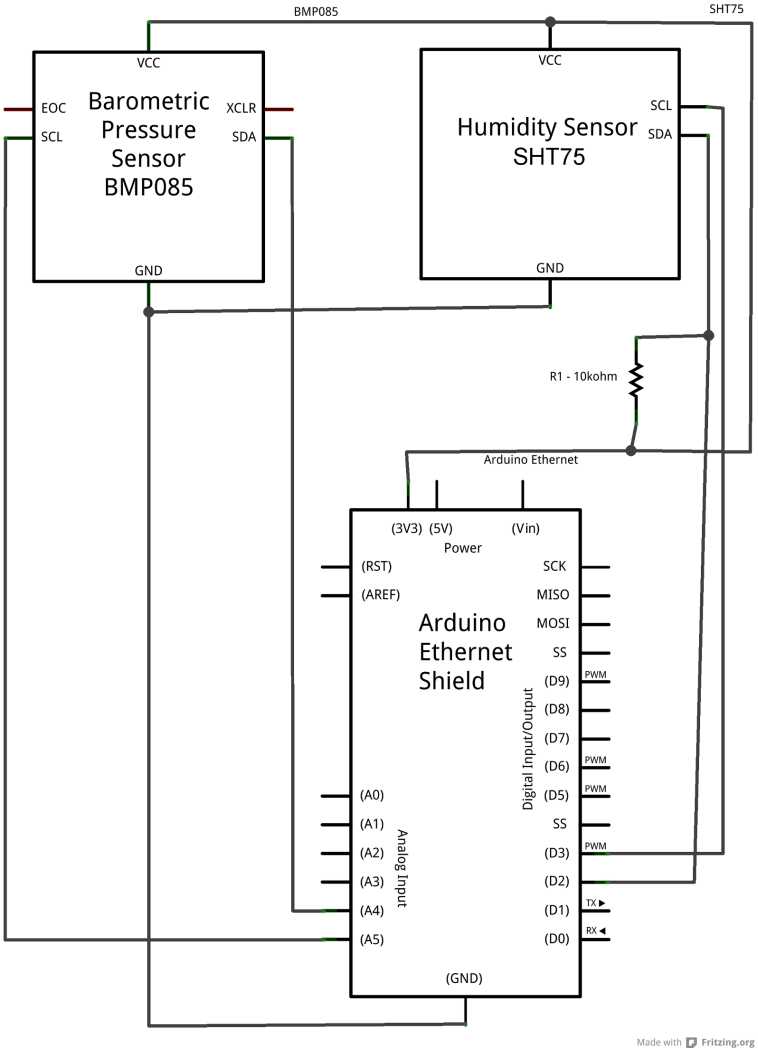
Izdelava vremenske postaje

4.1 Povezava senzorjev na mikrokrmilnik Arduino

Pred izdelavo vremenske postaje oziroma priklopom senzorjev na mikrokrmilnik Arduino, smo naredili shemo, ki jo prikazuje slika 4.1. S shemo smo si pomagali pri fizičnem priklopu senzorjev na prototipni plošči (angl. Breadboard, Protoboard). Prototipna plošča je po navadi uporabljena, ko želimo preizkusiti delovanje vezja, brez da bi vse komponente fiksno združili. Pri uporabi prototipne plošče komponente povežemo oziroma nataknemo na ploščo, vezi med njimi pa začasno naredimo s tankimi srednje trdimi kabli.
Povezovanje senzorja BMP085 je bilo precej enostavnejše kot povezovanje senzorja SHT75. Kot vidimo na sliki 4.2, je senzor BMP085 na evalvacijski plošči. To pomeni, da je senzor relativno enostavno priklopiti, saj ima na sami plošči že priklope, ki so enakih dimenzij kot priklopi na prototipni plošči. Prav tako je prednost tudi to, da so na evalvacijski plošči že naspajkani 10 kΩ upori pull-up, ki bi jih morali drugače sami povezati zaradi uporabe I^2C vodila.

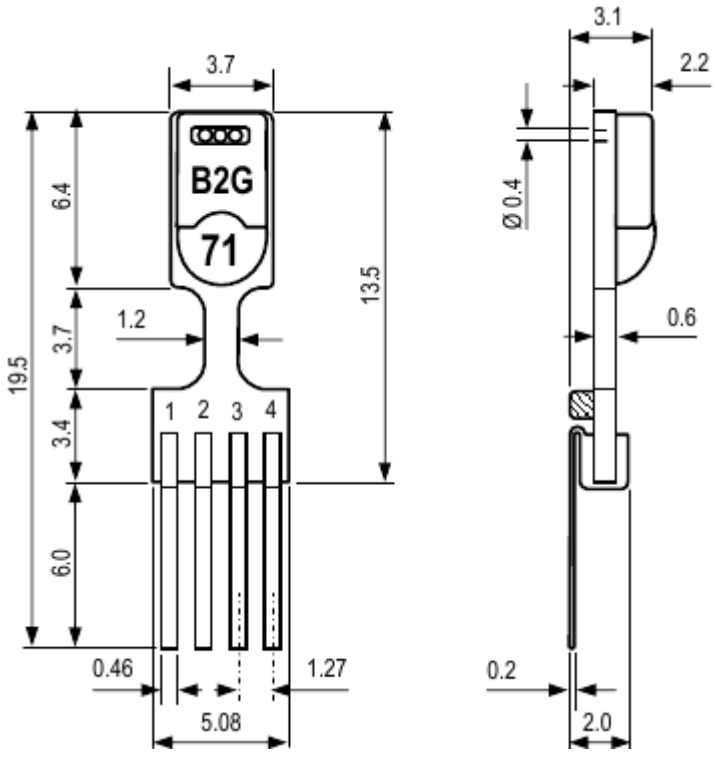
Senzor SHT75 je bil trši oreh, saj ni bil na plošči. Prav tako so bili razmaki med nogicami ter same nogice manjše kot na prototipni plošči, kot nam prikazuje slika 4.3. To smo rešili tako, da smo na nogice naspajkali žice, ki smo jih uporabili za povezovanje na testno ploščico. Tudi spajkanje je bilo zaradi dimenzij kar zahtevno, a nam je z mirno roko uspelo. Pri povezavi senzorja je prav tako potrebno povezati

Slika 4.1: Shema fizičnega priklopa senzorjev na mikrokrmilnik Arduino



Slika 4.2: Slika senzorja BMP085 na evalvacijski ploščici ATAVRSBPR1[29]

Slika 4.3: Dimenzije senzorja SHT75[16]



SDA ter VCC nožico preko 10kΩ upora (upor pull-up).

4.2 Program za mikrokrmilnik Arduino

Pri razvoju programa, ki bo tekel na mikrokrmilniku Arduino, smo uporabljali uradno programsko opremo za prevajanje ter nalaganje programa, za pisanje knjižnic ter samega programa pa urejevalnik Kate [30]. Urejevalnik Kate smo uporabili, ker uradna programska oprema Arduino ne podpira nekaterih naprednejših funkcij, na primer predlaganja vnosa ter zlaganja kode.
Diagram poteka glavnega programa, ki teče na mikrokrmilniku, prikazuje slika 4.4. Iz slike 4.4 lahko potegnemo vzporednici na dva glavna dela Arduino programa, to sta funkciji

- `void setup()` ter
- `void loop()`.

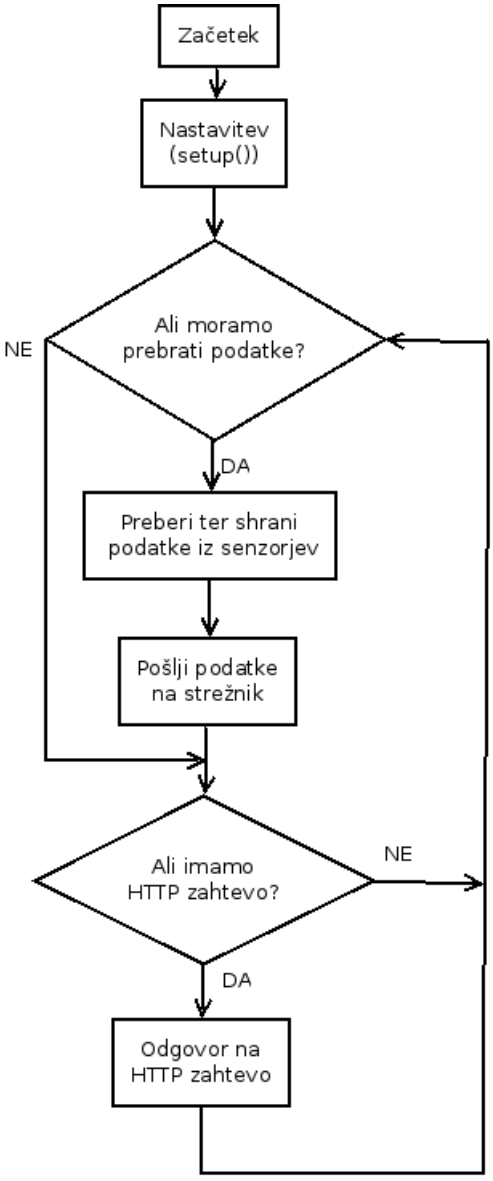
Na mikrokrmilniku Arduino se funkcija `void setup()` izvede natanko enkrat na začetku programa, za tem pa se do izklopa mikrokrmilnika izvaja funkcija `void loop()`. V funkciji `void setup()` smo nastavili naslednje parametre:

- Inicializacija senzorja BMP085,
- Inicializacija Ethernet vmesnika,
- Inicializacija HTTP strežnika,
- Inicializacija serijske povezave za pomoč pri razhroščevanju.

Funkcija `void loop()` ob izvajanju programa opravlja naslednje naloge:

1. Vsakih n minut (privzeto $n = 10$) prebere podatke iz senzorjev, jih zapiše v globalne spremenljivke ter pošlje na strežnik za zbiranje podatkov;
2. Preveri, če imamo novo zahtevo HTTP, ter ji ustrezno postreže.

Slika 4.4: Potek glavnega programa na mikrokrmilniku Arduino



4.2.1 Komunikacija s senzorji

Za komunikacijo s senzorji smo se odločili za dva pristopa:

- Komunikacija s senzorjem BMP085: Napisali smo svojo knjižnico, poimenovano `BMP085`;
- Komunikacija s senzorjem SHT75: Uporabili smo knjižnico `Sensirion` [31].

Knjižnica Sensirion

Knjižnica `Sensirion` nam omogoča komunikacijo s senzorji `SHT1x` ter `SHT7x` [31] proizvajalca Sensirion. Komunikacija s senzorjem `SHT75` poteka preko sinhronega digitalnega vodila prek dveh žic: `DATA` ter `CLOCK` [16](prek ene se prenašajo podatki, prek druge pa sinhrono ura). Pri odločitvi za uporabo knjižnice `Sensirion` je bil glavni razlog prav gotovo neuporaba vodila I^2C, prek katerega bi veliko lažje pridobili podatke iz senzorja.
Javni vmesnik API knjižnice `Sensirion` nam med drugim omogoča:

- Branje podatkov o temperaturi, vlažnosti;
- Izračun rosišča;
- Sinhrone ter asinhrone klice za pridobivanje podatkov;
- Izbiro resolucije merjenja podatkov (14/12 bit za temperaturo oz. 12/8 bit za vlažnost).

Pri naši uporabi smo uporabljali sinhrone klice pridobivanja podatkov ter višjo resolucijo merjenja. Pri izbiri višje resolucije se moramo zavedati, da je pridobivanje podatkov nekoliko daljše kot pri uporabi navadne resolucije. Pri uporabi visoke resolucije je čas za branje temperature ter vlažnosti 400 ms, pri uporabi manjše resolucije pa 100 ms [16]. Prav tako moramo paziti, da ne pregrejemo senzorja, kar se lahko zgodi ob preveč pogostem branju vrednosti. Priporočljivo je, da je senzor v uporabi samo 10 % časa [16].

Slika 4.5: Kalibracijske spremenljivke na senzorju BMP085 [17]

AC1 (0xAA, 0xAB)	(16 bit)
AC2 (0xAC, 0xAD)	(16 bit)
AC3 (0xAE, 0xAF)	(16 bit)
AC4 (0xB0, 0xB1)	(16 bit)
AC5 (0xB2, 0xB3)	(16 bit)
AC6 (0xB4, 0xB5)	(16 bit)
B1 (0xB6, 0xB7)	(16 bit)
B2 (0xB8, 0xB9)	(16 bit)
MB (0xBa, 0xBB)	(16 bit)
MC (0xBC, 0xBD)	(16 bit)
MD (0xBE, 0xBF)	(16 bit)

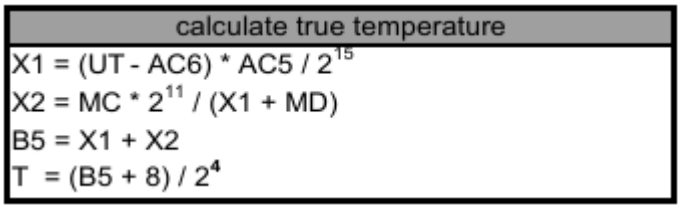
Knjižnica BMP085

Za komunikacijo s senzorjem BMP085 smo se odločili narediti knjižnico, ki smo jo poimenovali `BMP085`. Knjižnica ima naslednje metode:

- `BMP085::BMP085(SensorAddress)`,
- `BMP085::Calibration()`,
- `BMP085::ReadTemperature()`,
- `BMP085::ReadPressure()`.

Pri izdelavi knjižnice smo uporabljali tehnični priročnik senzorja [17], v katerem so natančno zapisani vsi potrebni koraki za branje podatkov.
Preden lahko uporabljamo senzor, moramo klicati metodo `BMP085::Calibration()`, ki iz senzorja prebere kalibracijske podatke prikazane na sliki 4.5, ter jih zapiše v zasebne spremenljivke knjižnice. Prav tako si knjižnica ob klicu konstruktorja (`BMP085::BMP085(SensorAddress)`) zapomni I^2C naslov senzorja (privzet je `0x77`). Ti podatki so uporabljeni pri računanju podatkov o temperaturi ter pritiska. Posebnost senzorja BMP085 je ta, da ne moremo brati samo podatkov o samo eni vrednosti (npr. samo temperature ali samo pritiska). Pri vsakem merjenju moramo torej prebrati najprej temperaturo ter nato pritisk.

Slika 4.6: Izračun temperature z uporabo nekompenzirane vrednosti, pridobljene iz senzorja BMP085 [17]



Pri merjenju temperature ter pritiska moramo iz senzorja najprej pridobiti nekompenzirano vrednost. Za pridobitev nekompenzirane vrednosti najprej v senzor pošljemo zahtevo za meritev temperature ali pritiska. Nato v kontrolni register na naslovu `0xF4` zapišemo:

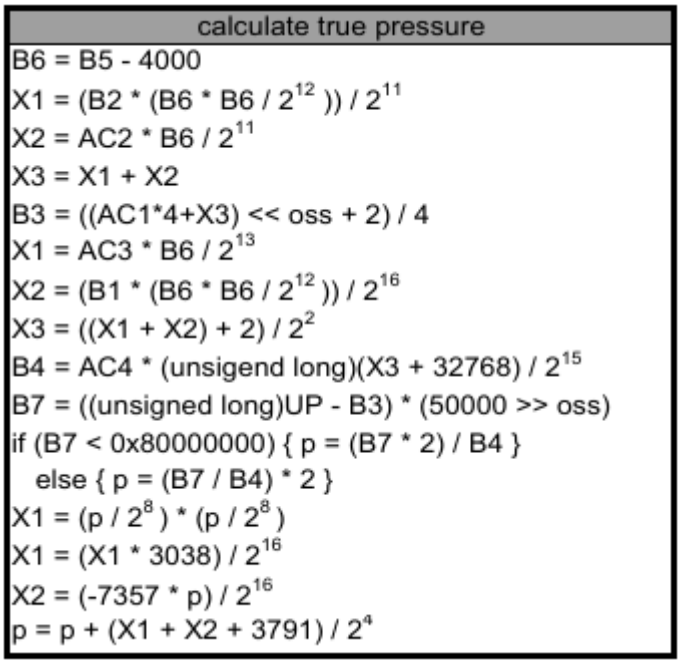
- `0x2E` pri branju temperature;
- `0x34 + (OSS<<6)` pri branju pritiska. Vrednost `OSS` je nastavitev resolucije branja (angl. OverSampling Setting), ki je lahko (0..3), kar pomeni uporabo 1, 2, 4 ali 8 vzorcev pri izračunu pritiska.

Ko smo pridobili nekompenzirane vrednosti iz senzorja, smo jih morali pretvoriti v dejanske vrednosti temperature (v °C) ter pritiska (v Pa). Formule za pretvorbo so bile podane v tehničnem priročniku senzorja, prikazujeta pa jih sliki 4.6 ter 4.7.

4.2.2 Odgovor na zahteve HTTP

Arduino programska oprema nam omogoča poganjanje strežnika na poljubnih vratih z objektom `EthernetServer`. Kot nam prikazuje slika 4.4, potem ko preverimo, ali moramo poslati podatke na strežnik ter ali moramo prebrati podatke iz senzorjev, preverimo, ali smo mogoče dobili novo povezavo. Funkcija `EthernetServer.available()` nam v primeru novega odjemalca vrne objekt `EthernetClient`, prek katerega beremo zahteve. `EthernetClient` nam omogoča branje podatkov znak za znakom s funkcijo `EthernetClient.read()`. Za potrebe diplomske naloge smo si vsako prebrano vrstico shranjevali v začasno spremenljivko,

Slika 4.7: Izračun pritiska z uporabo nekompenzirane, vrednosti pridobljene iz senzorja BMP085 [17]



Slika 4.8: Primer HTTP zahteve

```
(1) GET / HTTP/1.1
(2) Host: postaja
(3) User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:22.0)
    Gecko/20100101 Firefox/22.0
(4) Accept: text/html,application/xhtml+xml,
            application/xml;q=0.9,*/*;q=0.8
(5) Accept-Language: en-US,en;q=0.5
(6) Accept-Encoding: gzip, deflate
(7) Connection: keep-alive
(8) Cache-Control: max-age=0
```

prek katere smo potem ugotavljali, ali odjemalec želi navadno stran HTML ali podatke kodirane v formatu JSON. Za odgovor na zahtevo smo uporabili isti `EthernetClient` objekt, v katerega smo preko funkcij `EthernetClient.print()` ter `EthernetClient.println()` pisali standardni odgovor HTTP. Primer zahteve HTTP iz brskalnika prikazuje slika 4.8. Iz zahteve HTTP prikazane na sliki 4.8 lahko razberemo:

- Zahteva za podatke v obliki HTML (`1`): v primeru zahteve za podatke v obliki JSON, bi se zahteva začela z `GET /json` ;
- Naslov URL, v katerega se je razrešil naslov IP, na katerem se nahaja vremenska postaja, je "postaja" (`2`);
- Spletni odjemalec je spletni brskalnik Firefox na operacijskem sistemu Linux (`3`);
- Odjemalec bo najraje sprejel podatke v obliki HTML, XHTML ali XML (`4`), angleškega jezika (`5`), lahko tudi stisnjene z uporabo algoritmov `gzip` ali `deflate` (`6`);
- Odjemalec želi, da se povezava ohrani aktivna (`7`) ter

Slika 4.9: Primer HTTP odgovora

```
HTTP/1.1 200 OK
Content-Type: text/html
Connection: close
<!DOCTYPE HTML>
<html>
<head>
<title>Arduino Weather Station</title>
</head>
<body>
<h1>Arduino Weather Station</h1>
<h2>Current weather data</h2>
Temperature: 24.14°C<br/>
Pressure: 954.11hPa<br/>
Humidity: 60.56%<br/>
Dew: 16.03°C<br/>
</body>
</html>
```

- Odjemalec želi najnovejše podatke (8).

Primer odgovora vremenske postaje na zahtevo, prikazano na sliki 4.8, vidimo na sliki 4.9.

Veliko večino podatkov v zahtevi strežnik zaradi preproste arhitekture ignorira. Strežnik na vremenski postaji smo sprogramirali tako, da zna samo razlikovati med tem, ali odjemalec želi podatke v obliki HTML ali JSON.
Pri pisanju HTTP odgovora moramo upoštevati nekatere obvezne dele HTTP protokola. To pomeni, da moramo, preden začnemo pošiljati dejansko HTML stran ali json podatke, poslati HTTP zaglavje. V HTTP zaglavju je zapisano:

- Ali je bila zahteva uspešno postrežena;

Slika 4.10: Primer HTTP zaglavja ob uspešni zahtevi

```
(1) HTTP/1.1 200 OK
(2) Content-Type: text/html
(3) Connection: close
```

- Vrsta podatkov;
- Kaj naj odjemalec naredi s povezavo.

Ob uspešni zahtevi postaja vrne zaglavje, prikazano na sliki 4.10, iz katerega odjemalec razbere:

- Da je bila zahteva uspešno obdelana (`1`);
- Da so podatki tipa HTML (ob zahtevi za podatke JSON je ta vrstica `Content-Type: application/json`) (`2`);
- Da naj odjemalec zapre povezavo (`3`).

4.2.3 Pošiljanje podatkov na strežnik

Za pošiljanje podatkov v podatkovno bazo uporabljamo programski vmesnik, opisan v poglavju 5.4. Vremenska postaja ima v program zapisan strežnik ter naslov URL vmesnika, prek katerega pošilja podatke. Za pošiljanje smo uporabili objekt `EthernetClient`, ki smo ga že uporabljali pri odgovoru na HTTP zahteve (podpoglavje 4.2.2). Razlika med prejšnjo in zdajšnjo uporabo je bila v tem, da smo se sedaj sami povezali na strežnik, ter najprej pisali in nato brali iz objekta (oziroma strežnika). Kot tudi pri branju zahtev, smo morali paziti, da smo uporabljali standardne HTTP zahteve, ki jih bo spletni strežnik razumel. HTTP zahteva, ki jo je poslala vremenska postaja, je prikazana na sliki 4.11 (znak \ predstavlja nadaljevanje vrstice).

Pri zahtevi smo morali paziti na:

Slika 4.11: Primer HTTP zahteve ob klicu vmesnika API za pošiljanje podatkov

```
(1) GET /api/put/values.php\
?key=9f0259c6f44f1f22d43a017f2363e4fd2e9e94612f24749fc1da975de13e0b35\
&temperature=28.4\
&humidity=29\
&pressure=1015\
&dew=28.4 HTTP/1.1
(2) Host: vreme.repinc.eu
(3) Connection: close
```

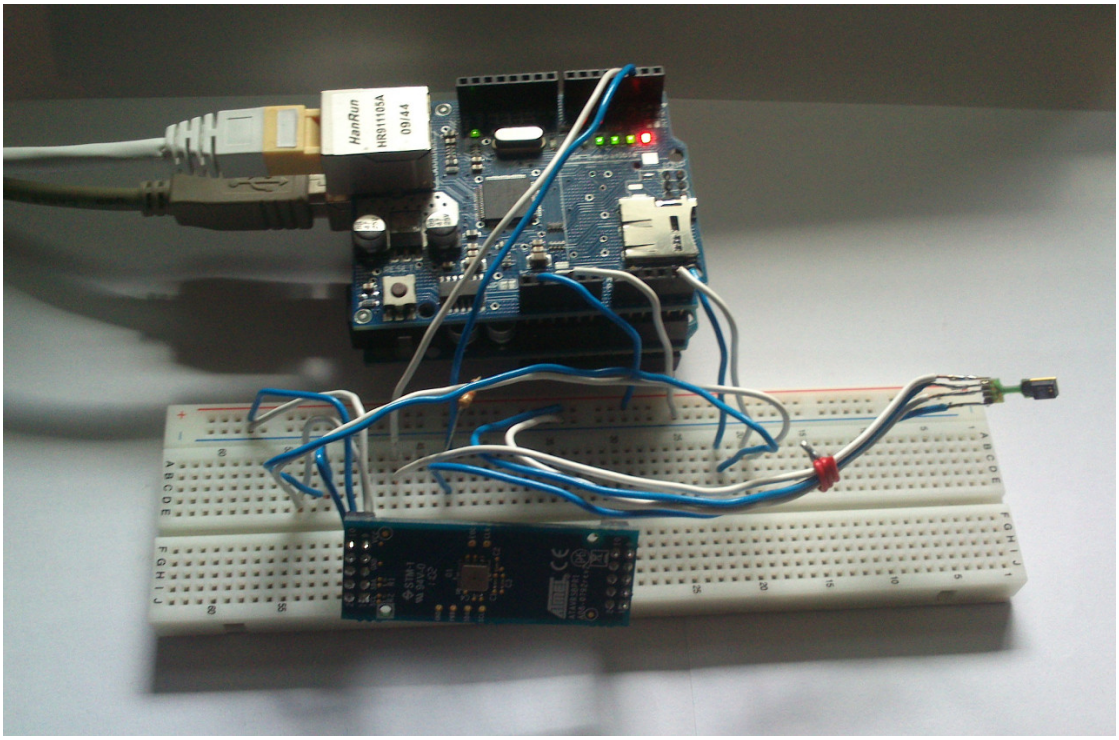
- Pravilno sintakso HTTP zahteve GET (`1`):
  - `GET x HTTP/1.1`: zahteva GET mora biti brez naslova strežnika, dodana mora biti specifikacija različice protokola HTTP, ki ga uporabljamo;
  - Parametri GET se začnejo po naslovu URL z znakom `?`, med vsakim parametrom mora biti znak `&`.
- Pri zahtevi smo morali dodati vrstico z imenom gostitelja (`2`), saj na enih vratih lahko posluša strežnik, ki zna streči več strani glede na ime gostitelja (npr. `www.gostitelj.si` ter `gostitelj.si` lahko vračata različne spletne strani);
- Povedali smo tudi, naj se povezava zapre, saj naš program ni narejen tako, da bi povezavo HTTP vzdrževal.

Ob uspešni zahtevi vmesnik API vrne potrditev, ki pa jo v našem programu na vremenski postaji nismo preverjali.

4.3 Končna oblika vremenske postaje

Vremensko postajo prikazuje slika 4.12, na kateri vidimo uporabljene senzorje na prototipni plošči, ščit Ethernet, pod njim pa mikrokrmilnik Arduino Uno. Vse povezave senzorjev so začasne, narejene so z modrimi in belimi kabli.

Slika 4.12: Vremenska postaja



Vremensko postajo lahko v obliki, prikazani na sliki 4.12, uporabljamo za merjenje notranjih vremenskih podatkov. Za uporabo v zunanjem okolju bi morali senzorje in mikrokrmilnik ustrezno zaščititi pred zunanjimi vplivi.

Poglavje 5

Izdelava programske opreme

Pri izdelavi programske opreme smo uporabili tehnologije in komponente opisane v poglavju 3.2. V trenutni različici programske opreme je podprto shranjevanje in pridobivanje podatkov o temperaturi, pritisku, vlažnosti ter rosišču.

5.1 Struktura spletne strani

Slika 5.1 nam prikazuje strukturo datotek na datotečnem sistemu. Iz nje je razvidna dokaj klasična struktura spletne strani. V vrhnji mapi sta datoteki `index.php` ter `config.php`, ki predstavljata uporabniško spletno stran, ki je prikazana na zaslonskem posnetku 5.2. V datoteki `config.php` lahko uporabnik nastavlja naslov spletne strani in lokacijo, ta podatka se uprabita v naslovu in podnaslovu strani, prav tako pa tudi v naslovu HTML dokumenta.

5.2 CSS

Datoteka `css/style.css` predstavlja stilsko predlogo, ki se uporablja za oblikovanje spletne strani. Spletna stran poizkuša biti čim bolj uporabniku prijazna ter enostavna za uporabo. Obliko lahko preprosto ponazorimo s tabelo 5.1. V tabeli vidimo, da je na vrhu čez celotno stran levo poravnan naslov, spodaj pa sta dva

Slika 5.1: Struktura datotek na datotečnem sistemu

```
| index.php
| config.php
|- api
   | info.php
   | db.config.php
   |- data
     | current.php
     | temperature.php
     | humidity.php
     | pressure.php
   |- put
     | values.php
     | station.php
   |- lib
     | database.php
     | data.php
   |- util
     | converter.php
     | random.php
|- css
  | style.css
|- img
  | ajax-loader.gif
|- js
  | arduinows.js
  | jquery.js
  |- highcharts
```

Slika 5.2: Zaslonski posnetek spletne strani ArduinoWS, Maj 2013

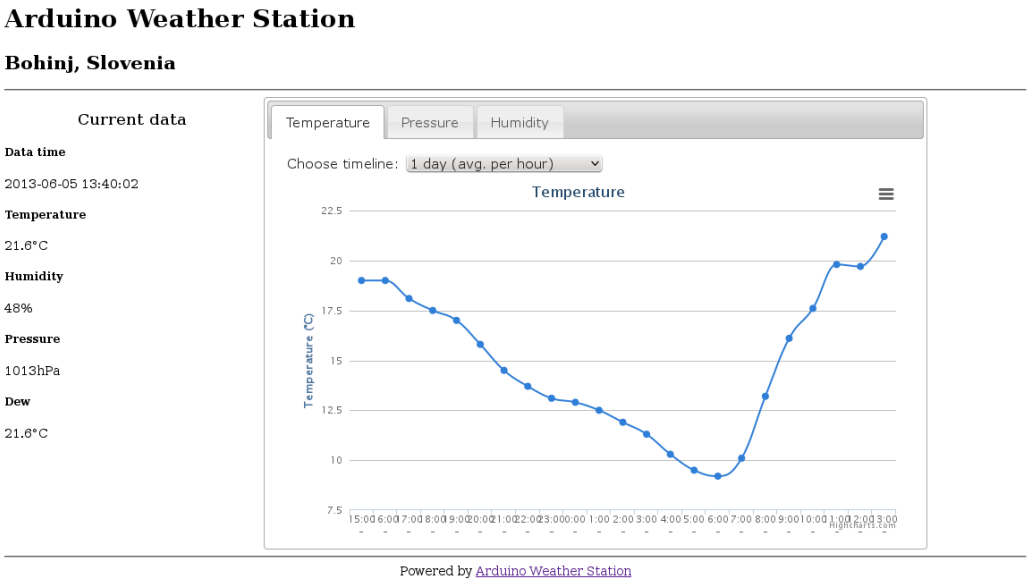


Tabela 5.1: Logična stilska postavitev spletne strani

Naslov	
Trenutni podatki	Grafi temperature, vlažnosti, pritiska

stolpca, ki imata razmerje približno 1 : 2.5.

5.3 Javascript

Ker spletna stran zelo močno sloni na uporabi programskega jezika Javascript, so knjižnice in skripte, ki smo jih uporabili pri izdelavi spletne strani, v podmapi `js/`. Uporabili smo dve zunanji knjižnici:

1. `jQuery`: Knjižnica, ki nam poenostavi prikazovanje različnih elementov spletne strani ter olajša delo z zahtevami AJAX;

2. `jQueryUI`: Knjižnica, ki smo jo uporabili za izdelavo nekaterih elementov spletne strani. Konkretno smo to knjižnico uporabili za izdelavo zavihkov,

ki jih uporabljamo za prikaz grafov temperature, vlažnosti ter pritiska.

3. `Highcharts`: Knjižnica, uporabljena za izris grafov temperature, vlažnosti ter pritiska.

Koda, v kateri se skriva funkcionalnost spletne strani, ter uporablja zgornje knjižnice, je v datoteki `js/arduinows.js`. Ta datoteka uporablja `jQuery` za manipuliranje spletne strani ter odzive na dogodke (npr. izbira časovnega okvirja podatkov pri izrisu grafa). Skripta `arduinows.js` je torej odgovorna za:

- Izpis trenutnih podatkov v levi stolpec spletne strani. To dosežemo tako, da del `jQuery` skripte pokliče `api/data/current.php` in s tem pridobi podatke. S temi podatki nastavi določeno vrednost `<div>` elementom spletne strani.
- Ob prikazu spletne strani ali uporabnikove izbire časovnega okvirja podatkov se osvežijo podatki v grafu. Pokliče se del skripte, ki ugotovi, kateri časovni okvir je bil izbran, in nato glede na to, katere vrste podatki so bili izbrani (temperatura, pritisk, vlažnost), skripta pridobi podatke iz določenega dela vmesnika `API` (npr. `api/data/temperature.php`). Ti podatki so uporabljeni za inicializacijo `Highcharts` grafa, ki omogoča preprost in lep vizualen pregled podatkov.

5.4 API

`API` (ang. Application Program Interface) je vmesnik, ki omogoča prenos podatkov med različnimi programi oziroma v našem primeru med različnimi komponentami vremenske postaje. Kot nam prikazuje slika 3.3, vse poti do podatkov vodijo prek vmesnika `API`. To smo naredili predvsem zato, da ločimo podatkovni del od prezentacijskega dela. Tudi s tem, ko za vse interakcije s podatkovno bazo uporabljamo eno mesto, poenostavimo cel sistem, in se potencialno izognemo velikim spremembam v delih kode, ki uporabljajo `API`. Na datotečnem sistemu je `API` v poddirektoriju `api/`.

`API` je napisan v programskem jeziku PHP, za pošiljanje podatkov uporablja standardno HTTP metodo `GET`, podatke pa vrača v obliki `JSON`. Slika 5.3 nam prikazuje

Slika 5.3: Tipičen API klic

```
GET api/data/temperature.php?range=day&fahr=true HTTP/1.1
```

Slika 5.4: Tipični JSON kodirani podatki, ki jih vrne vmesnik API

```
{
  "from": [0,1,2,3,15,16,17,18,19,20,21,22,23],
  "to":   [1,2,3,4,16,17,18,19,20,21,22,23,24],
  "data": [57.5,55.9,53.1,51.8,66.2,66.3,64.5,
   63.5,62.6,60.4,58.1,56.6,55.6]
}
```

tipičen API klic, v tem primeru želimo pridobiti podatke o temperaturi v zadnjem dnevu (povprečno po urah, specifikacija v poglavju 5.4.1), temperature pa naj bodo pretvorjene v °F.

Tak klic nam vrne podatke v formatu `JSON`, kot vidimo na sliki 5.4 (za potrebe prikaza smo izpis "olepšali", dejanski podatki so vrnjeni brez novih vrstic in odvečnih presledkov). Sam vmesnik ima konfiguracijsko datoteko `api/db.config.php`, v kateri so shranjeni konfiguracijski podatki podatkovne baze. Prav tako ima vmesnik še klic `api/info.php`, ki nam vrne podatke o različici vmesnika, kot nam prikazuje slika 5.5. Ta klic je uporaben predvsem, ko želimo ugotoviti, katera različica vmesnika se uporablja na strežniku, saj je lahko z različico pogojena funkcionalnost vmesnika API. Različne dele vmesnika API bomo predstavili v svojih podpoglavjih, saj so obsežnejša.

Slika 5.5: Podatki o vmesniku API, ki jih pridobimo s klicem api/info.php

```
{
  "name":          "ArduinoWS API",
  "description":   "Arduino Weather Station API",
  "version":       "0.1",
  "version_major": 0
  "version_minor": 1
  "version_patch": 0
  "url":            "https://github.com/jernejovc/arduinows/"
}
```

5.4.1 Časovni okvir podatkov

Za pridobivanje podatkov prek metode `api/data/` je potrebno ob vsakem klicu navesti časovni okvir podatkov, ki jih želimo pridobiti. Z uporabo GET spremenljivke `range` nastavljamo željen časovni okvir pri klicu. Tabela 5.2 nam prikazuje veljavne vrednosti. Vmesnik trenutno ne podpira poljubnega časovnega okvirja in izbire povprečja po dnevih/urah.

5.4.2 api/data

`api/data/` del vmesnika je namenjen pridobivanju podatkov iz baze. V različici, ki je bila razvita za potrebe diplomske naloge, smo implementirali pridobivanje naslednjih podatkov:

- Temperatura: `api/data/temperature.php`, klic lahko nastavi parameter `fahr` na vrednost `true`, s tem so vrnjene temperature v enotah °F namesto v °C, kot privzeto.
- Pritisk: `api/data/pressure.php`, vrednosti so v enotah hPa (mbar).
- Vlažnost: `api/data/humidity.php`, vrednosti so v enotah % relativne vlažnosti.

Tabela 5.2: Veljavne vrednosti časovnega okvirja pri uporabi api/data/

Vrednost	Časovni okvir
`dayfull`	En dan, vsi podatki.
`day`	En dan, povprečni podatki na uro.
`week`	En teden, povprečni podatki na uro.
`month`	En mesec, povprečni podatki na dan.
`3months`	Trije meseci, povprečni podatki na dan.
`year`	Eno leto, povprečni podatki na dan.
`all`	Vsi podatki, povprečni podatki na dan.

Klici vseh teh metod morajo imeti dodan parameter `range`, ki smo ga opisali predhodno (poglavje 5.4.1). Vsi klici metod vračajo podatke v formatu `JSON`, katerega izgled je prikazan na sliki 5.4. Za vrnjene podatke je značilno:

- Podatki so razdeljeni na več polj, odvisno od časovnega okvirja podatkov,
- Dolžine vrnjenih polj so identične (npr. $length(data) == length(time)$), torej za vsak $i \in$ `[0..length(data)]` $\exists$ `time[i]`,
- Polje `time` prikazuje časovni okvir vrnjenih podatkov (npr. točen čas podatka, ura, dan, teden v letu).
- Polje `data` prikazuje podatke pri določeni časovni enoti.

Glede na zahtevan časovni okvir so vrnjenim podatkom dodani metapodatki o času, v katerem so bili podatki izmerjeni oziroma izračunani:

- `dayfull`
  - `time`: Čas, ob katerem so bili podatki izmerjeni (npr. `12:34:42`)
- `day`
  - `from`: Ura, od katere so bili podatki vzeti (npr. `12`).

  - `to`: Ura, do katere so bili podatki vzeti (npr. `13`).
  - `time`: Ura od in do katere so bili podatki vzeti (npr. `12:00 - 13:00`, kombinacija zgornjih polj)

- `week`
  - `from`: Ura, od katere so bili podatki vzeti (npr. `12`).
  - `to`: Ura, do katere so bili podatki vzeti (npr. `13`).
  - `time`: Dan ter ura od in do katere so bili podatki vzeti (npr. `2013-1-1 12:00 - 13:00`, kombinacija zgornjih polj)

- `month`
  - `time`: Dan, na katerega so bili podatki izmerjeni (npr. `2013-1-1`).

- `3months`
  - `time`: Dan, na katerega so bili podatki izmerjeni (npr. `2013-1-1`).

- `year`
  - `time`: Teden in leto, v katerem so bili podatki izmerjeni (npr. `1/2013`).

- `all`
  - `time`: Čas, ob katerem so bili podatki izmerjeni (npr. `12:34:42`)

Kot je razvidno iz zgornjega seznama, klici z vsakim časovnim okvirjem vrnejo vsaj dve polji, `data` ter `time`. S tem smo tekom razvoja vmesnika poizkusili letega narediti čim bolj konsistentnega za uporabnike ter razvijalce. Določeni deli vmesnika vrnejo še dodatne metapodatke, na primer `day` ter `week` vrneta še polji `from` ter `to`. Pri izdelavi oziroma načrtovanju dela vmesnika `api/data` smo se zaenkrat osredotočili le na glavne potrebe pri pridobivanju podatkov (glede na Paretovo načelo[32], 80 % funkcionalnosti smo naredili v 20 % časa). Zaenkrat za pridobivanje podatkov uporabnik ne potrebuje ključa, kot na primer za dodajanje podatkov. To z varnostnega stališča ni priporočljivo, saj lahko hitro pride do

zlorab. V naslednjih različicah programskega vmesnika in spletne strani je predviden uporabniški vmesnik, v katerem bo administrator spletne strani lahko določil funkcionalnost uporabe ključa ter dodajal in odstranjeval uporabnike.

5.4.3 api/put

Del vmesnika `api/put` je namenjem dodajanju podatkov iz vremenskih postaj v podatkovno bazo ter dodajanju vremenskih postaj v sistem. Da lahko vremenska postaja doda podatke v podatkovno bazo, mora najprej dobiti ključ, s katerim se predstavi vmesniku. Ključ je naključno generiran niz znakov, kodiran z zgoščevalno metodo SHA-256. Zgoščevalna metoda SHA-256 niz katerekoli dolžine zgosti v unikatno 256-bitno vrednost. Lastnost zgoščevalne funkcije SHA-256 je tudi tako imenovani plazovni efekt (ang. Avalanche Effect). To pomeni, da že majhna sprememba v nizu pomeni, da je zgoščena vrednost čisto drugačna kot pri nespremenjenem nizu[33].

api/put/station.php

Za pridobitev ključa uporabnik pokliče del programskega vmesnika `api/put/station.php` z naslednjimi `GET` parametri (primer klica nam prikazuje slika 5.6:

- `name`: Ime vremenske postaje, obvezen parameter, do 50 črk.
- `description`: Opis vremenske postaje, obvezen parameter, do 200 črk.
- `location`: Lokacija vremenske postaje, neobvezen parameter, do 50 črk.

Če je zahteva uspela, nam programski vmesnik vrne kodirane podatke `JSON` s ključem dodane vremenske postaje, kot to prikazuje slika 5.7. V primeru neuspeha je odgovoru dodan element `error` z opisom napake.

Kot vidimo na sliki 5.7, nam ob uspehu programski vmesnik vrne še naše podatke o vremenski postaji, ki smo jih posredovali ob klicu.

Slika 5.6: Prikaz klica dela programskega vmesnika za dodajanje vremenske postaje

```
GET api/put/station.php?name=Test name&description=Test description\
&location=Test location HTTP/1.1
```

Slika 5.7: Prikaz uspešnega odgovora ob klicu dodajanja vremenske postaje

```
{
  "status":      "ok",
  "hash":        \
  "e72870b83905674a62ef3cfb55a93e179428917537ad9a39f9e1d2d35b4f107b",
  "name":        "Test name",
  "description": "Test description",
  "location":    "Test location"
}
```

api/put/values.php

Del vmesnika `api/put/values.php` je uporabljen za dodajanje vrednosti v podatkovno bazo. Kot smo povedali že na začetku poglavja 5.4.3, moramo pri vsakem klicu za dodajanje podatkov v bazo ob klicu dodati še generiran ključ, ki predstavlja točno določeno vremensko postajo. Preko vmesnika lahko trenutno v podatkovno bazo dodajamo vremenske podatke o:

- Temperaturi;
- Vlažnosti;
- Pritisku;
- Rosišču.

Ob klicu `api/put/values.php` moramo dodati naslednje GET parametre:

- `hash`: Veljaven SHA-256 ključ, ki enolično določa vremensko postajo;
- `temperature`: Vrednost temperature;
- `humidity`: Vrednost vlažnosti;
- `pressure`: Vrednost pritiska;
- `dew`: Vrednost rosišča.

Pri klicu so vsi zgornji parametri obvezni, kot vidimo na sliki 5.8. Ob uspešnem klicu nam vmesnik vrne kratko potrditev, da je vnos podatkov uspel, kar vidimo na sliki 5.9.

Ob klicu služi parameter `hash` kot unikatni identifikator vremenske postaje, ki se interno v podatkovni bazi pretvori v `id` vremenske postaje.

5.4.4 api/lib

V direktoriju `api/lib/` so shranjeni podporni razredi, ki jih uporabljajo deli vmesnika `api/data` ter `api/put`. Podporni razredi v poddirektoriju `api/lib` so

Slika 5.8: Primer zahteve za dodajanje podatkov v bazo

```
GET api/put/values.php
?key=9f0c59c6f44f1f22bb3a017f2363e4fd2aae94612f24749fc1da975de13e0bd5
&temperature=16.3&humidity=75&pressure=1014&dew=16.3 HTTP/1.1
```

Slika 5.9: Primer odgovora ob uspešnem dodajanju podatkov v bazo

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

večinoma uporabljeni za interakcijo s podatkovno bazo. Kot nam prikazuje slika 5.1, sta del teh direktorijev datoteki

- `api/lib/database.php` ter
- `api/lib/data.php`.

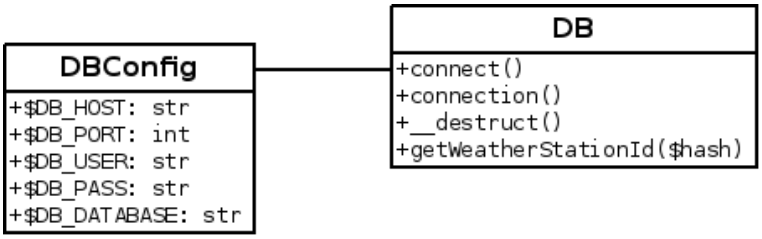
api/lib/database.php

V datoteki je razred DB, ki nam služi za povezovanje na podatkovno bazo. Vsak del vmesnika, ki potrebuje povezavo na podatkovno bazo, torej povezavo dobi preko tega razreda. Razred prebere podatke o nastavitvah podatkovne baze iz datoteke `api/db.config.php`. UML diagram, ki ga vidimo na sliki 5.10 nam prikazuje funkcije ter atribute razredov v prej omenjenih datotekah. Iz njega je razvidno, da razred DB uporablja podatke oziroma atribute razreda DBConfig.

api/lib/data.php

V datoteki je razred Data, s pomočjo katerega pridobivamo podatke iz podatkovne baze ter razreda RangeEnum ter KindEnum. Slika 5.11 nam prikazuje razrede v

Slika 5.10: UML diagram razredov v datotekah api/db.config.php ter api/lib/-database.php



datoteki `api/lib/data.php`. Kot vidimo ob vsakem klicu funkcije `DataFetcher.getData` dodamo še parameter tipa `KindEnum`, ki nam pove, kakšen tip podatkov želimo ter parameter tipa `RangeEnum`, s katerim nastavljamo časovni okvir podatkov. Prav tako ob klicu funkcije lahko nastavimo še parameter `$fahrenheit`, s katerim ob zahtevi za temperaturo le-to pretvorimo v stopinje Fahrenheita namesto v privzete stopinje Celzija.
Ta razred uporablja prej omenjeni razred DB (podpoglavje 5.4.4) za povezavo na podatkovno bazo.

5.4.5 api/util

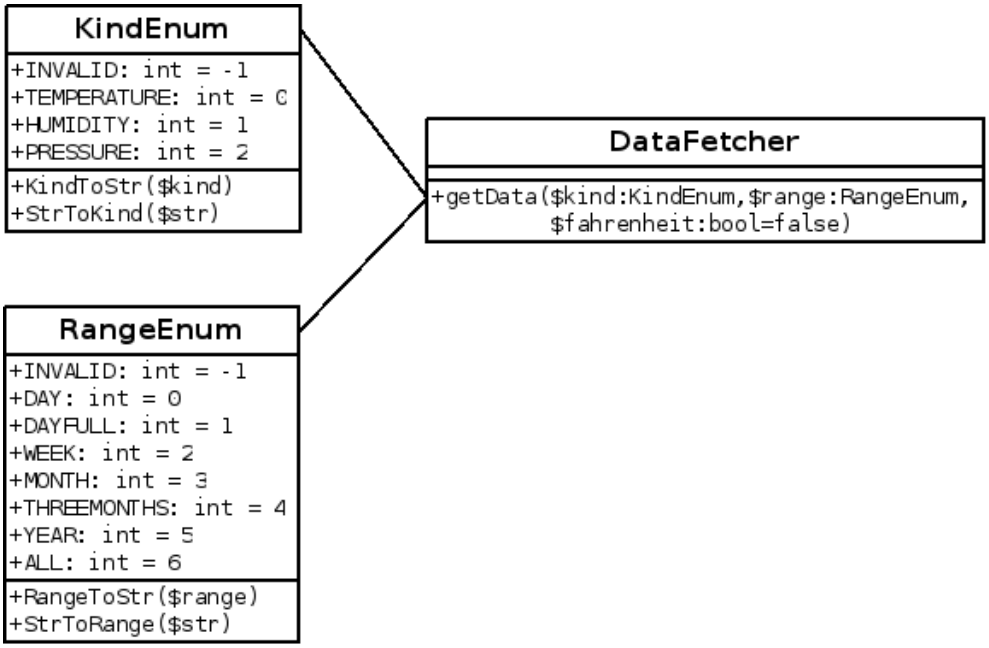
V poddirektoriju `api/util/` se nahajata datoteki:

- `api/util/converter.php` ter
- `api/util/random.php`.

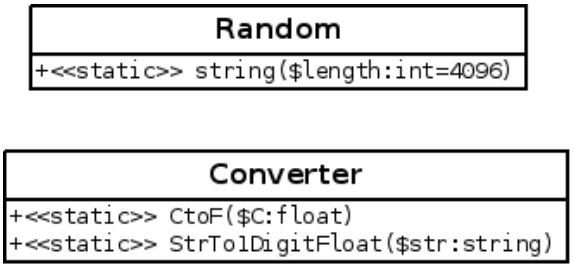
V teh datotekah najdemo razreda Converter ter Random, ki nam ju prikazuje slika 5.12. Na sliki vidimo, da so vse funkcije razredov statične, to pomeni, da ne potrebujemo narediti instance objekta za njihovo uporabo. Funkcije so uporabljene za:

- `Random.string`: Generiranje naključnega niza znakov določene dolžine, ki ga uporabljamo pri generiranju ključa vremenske postaje;
- `Converter.CtoF`: Pretvorba temperature iz stopinj Celzija v stopinje Fahrenheita;

Slika 5.11: UML diagram razredov v datoteki api/lib/data.php.



Slika 5.12: UML diagram razredov v podmapi api/util.



- `Converter.StrTo1DigitFloat`: Pretvorba niza znakov v float tip z 1 decimalnim mestom.

Pri izdelavi razredov ter načrtovanju funkcij smo si zadali cilj, da lahko kodo uporabimo na več različnih mestih.

5.5 Podatkovna baza

Načrt podatkovne baze prikazuje diagram ER (ang. Entity-Relationship diagram) na sliki 5.13. Na diagramu so prikazane tri entitete:

- `stations`;
- `weather_data`;
- `historical_data`.

Entiteta `weather_data` je prek zunanjega ključa iz `station` na `id` povezana s entiteto `stations`. V entiteto `weather_data` pišemo s pomočjo vmesnika `api/put/values.php`. V entiteto `historical_data` pa podatke pišemo preko MySQL Eventa, ki vsak dan iz tabele `weather_data` v entiteto `historical_data` premakne podatke, starejše od enega leta.
Entiteta `stations` ima naslednja polja:

- `id`: Identifikacijsko polje, tudi primarni ključ tabele. Polje je tipa `INT(11)`, `AUTO_INCREMENT` (kar pomeni, da se samodejno generira – načeloma je enak številu vstavljenih vrstic v entiteto). Vrednost mora biti unikatna (`UNIQUE`) v entiteti in polje ne sme biti prazno (`NOT NULL`).
- `hash`: Enolični identifikator postaje, uporabljen pri dodajanju podatkov. Polje je tipa `VARCHAR(64)` - `SHA256` vrača ključ dolg 256 bitov, to je 64 črk dolgo polje, ko je pretvorjeno v šestnajstiški številski sistem. Vrednost mora biti unikatna v entiteti, polje ne sme biti prazno.
- `name`: Ime vremenske postaje, poljuben niz znakov tipa `VARCHAR(50)`, torej dolžine največ 50 znakov. Polje ne sme biti prazno.

- `description`: Opis vremenske postaje, poljuben niz znakov tipa `VARCHAR(200)`, dolžine največ 200 znakov. Polje ne sme biti prazno.
- `location`: Informativna lokacija vremenske postaje, poljuben niz znakov tipa `VARCHAR(50)`, dolžine največ 50 znakov. Polje je lahko prazno.

Entiteta `weather_data` ima naslednja polja:

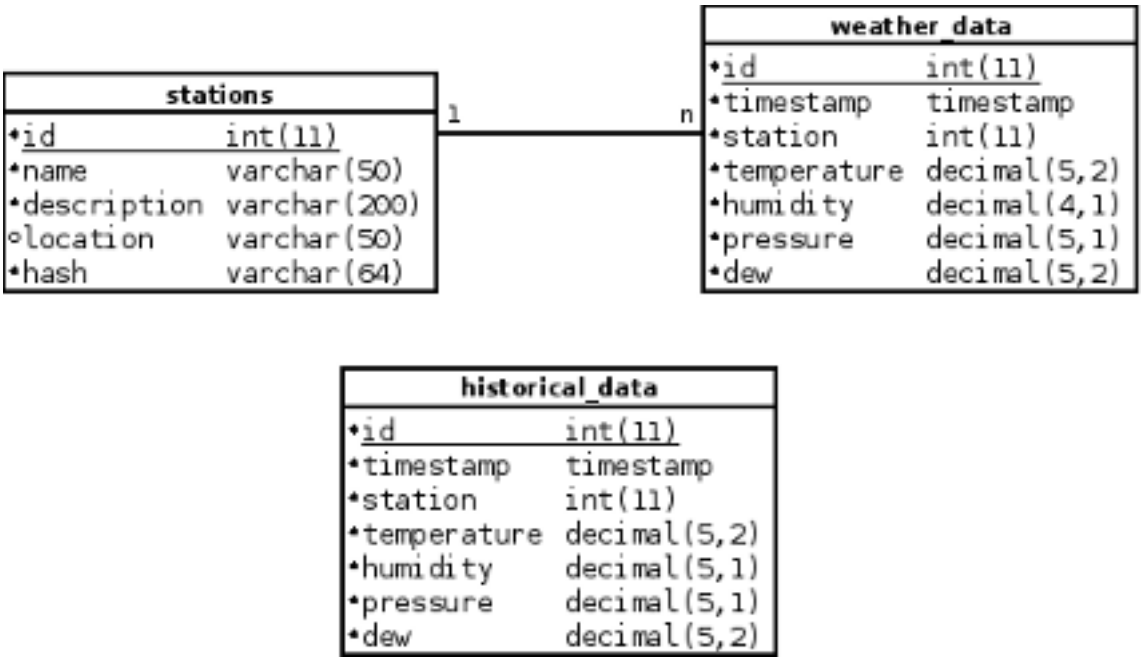
- `id`: Identifikacijsko polje, tudi primarni ključ tabele. Polje je tipa `INT(11)`, `AUTO_INCREMENT`. Vrednost mora biti unikatna (`UNIQUE`) v entiteti in polje ne sme biti prazno (`NOT NULL`).
- `timestamp`: Čas, ob katerem so bili podatki podani v bazo. Polje se samodejno generira, in sicer se ob vstavljanju doda, če ni določena vrednost trenutna vrednost MySQL funkcije `SYSDATE()`, ki vrne trenutni čas. Polje ne sme biti prazno.
- `station`: Zunanji ključ (angl. Foreign Key), ki nam pove, katera vremenska postaja je dodala podatke. Ta vrednost kaže na `id` polje v entiteti `stations`. Polje ne sme biti prazno.
- `temperature`: Vrednost temperature. Polje je tipa `DECIMAL(5,2)`, kar pomeni da je za zapis podatka namenjenih 5 mest, natančnost pa je 2 decimalki. To nam omogoča, da v tabelo zapisujemo vrednosti `[-999.99, 999.99]`.
- `humidity`: Vrednost vlažnosti. Polje je tipa `DECIMAL(4,1)`, omogoča nam zapis vrednosti `[-999.9, 999.9]`.
- `pressure`: Vrednost pritiska. Polje je tipa `DECIMAL(5,1)`, omogoča nam zapis vrednosti `[-9999.9, 9999.9]`.
- `dew`: Vrednost rosišča. Polje je tipa `DECIMAL(5,2)`, omogoča nam zapis vrednosti `[-999.99, 999.99]`.

Entiteta `historical_data` ima ista polja kot entiteta `weather_data`, s to razliko, da polje `timestamp` ni avtomatsko generirano, ampak se prenese iz entitete

`weather_data`.
Z omejevanjem vrednosti tako poskrbimo za manjšo velikost podatkovne baze, in že takoj poskrbimo za kolikor tolikšno preverjanje vrednosti ob vnosu v podatkovno bazo. Nad entiteto `stations` smo postavili indeks na vseh poljih. Ta pripomore k hitremu iskanju oziroma preverjanju vremenske postaje ali hash ključa. Na entiteti `weather_data` smo postavili indeks na polju `timestamp` zaradi večkratnega iskanja prav po tem polju. Na entiteto `historical_data` nismo postavili indeksov, saj pričakujemo, da ne bo uporabljena redno (do nje se zaenkrat ne da dostopati prek programskega vmesnika) oziroma bo namenjena shranjevanju velike količine podatkov. Zaradi velike količine podatkov bi ob dodajanju obremenili podatkovno bazo, saj bi morala generirati indekse.
Velikost podatkovne baze je sorazmerno majhna, če imamo vremensko postajo, ki vsakih 10 minut pošlje podatke v bazo, nam v enem letu zavzame približno:

- 1.3 MB čistih podatkov;
- 2.9 MB velik indeks;
- Skupaj 4.2 MB podatkov za eno postajo.

Slika 5.13: ER diagram podatkovne baze



Poglavje 6

Možne izboljšave

6.1 Strojne komponente

6.1.1 Mikrokrmilnik

Pri načrtovanju same vremenske postaje smo se omejili na mikrokrmilnik Arduino, predvsem zaradi ščita Ethernet (angl. Ethernet Shield), ki nam omogoča napajanje preko kabla Ethernet (angl. Power over Ethernet). Pri izdelavi postaje bi lahko uporabili tudi kakšen zmogljivejši mikrokrmilnik, ali pa katerega od majhnih računalnikov, na primer RaspberryPi[1] ali BeagleBone Black[2]. Ti računalniki v malem nam omogočajo veliko večjo zmogljivost ter prilagodljivost. Prav tako na njih lahko poganjamo operacijski sistem Linux, kar prinaša še večjo učinkovitost pri pisanju ter razhroščevanju programov. Morali bi sicer najti nek alternativni način napajanja oziroma pretvornik za napajanje preko kabla Ethernet. Prav tako bi lahko izdelali čisto avtonomno različico vremenske postaje s ščitom Arduino GSM (angl. Arduino GSM shield) preko katerega bi pošiljali podatke. Za napajanje bi lahko poskrbela kombinacija solarnih celic ter akumulatorja.

[1] `http://www.raspberrypi.org/`

[2] `http://beagleboard.org/Products/BeagleBone%20Black`

6.1.2 Senzorji

Trenutno smo v diplomski nalogi merili samo temperaturo, vlažnost ter pritisk. V prihodnosti bi morali dodati še senzorje za merjenje smeri in hitrosti vetra, količino padavin, moči sonca (koliko W/m^2 energije trenutno daje sonce) ... Izdelava oziroma nakup takih senzorjev pa bi bil relativno drag, zato smo se v okviru diplomske naloge odločili za uporabo omejenega števila senzorjev.

6.2 Programske komponente

V programskem delu vremenske postaje nam je ostalo še kar nekaj dela. Spletno stran bi lahko v nadaljnjem spremenili tako, da bi podpirala prikaz podatkov iz različnih vremenskih postaj. Prav tako ne obstaja vmesnik za nastavljanje podatkov o postajah, vmesnik za upravljanje z uporabniki, vmesnik nadzora dostopa. Programski vmesnik API bi moral doživeti razširitev v smislu nadzora dostopa, izbire podatkov, izdelave statističnih poročil.
Prav tako bi vmesnik API lahko razširili tako, da bi podpiral tudi druge podatkovne baze, na primer PostgreSQL ali povezavo na podatkovno bazo prek vmesnika ODBC.

Poglavje 7

Sklepne ugotovitve

V diplomski nalogi smo izdelali fizično vremensko postajo, ki beleži temperaturo zraka, vlažnost, pritisk ter rosišče. Vremenska postaja prek programskega vmesnika pošilja podatke v podatkovno bazo. Te podatke lahko pregledujemo prek uporabniku prijaznega spletnega vmesnika, ki uporablja programski vmesnik za dostop do podatkov. V prejšnjem poglavju o možnih izboljšavah smo izpostavili nekaj primerov, kako bi lahko samo vremensko postajo, spletno stran ter programski vmesnik izboljšali. Program, ki teče na mikrokrmilniku Arduino, bi lahko naredili še bolj modularen ter stabilen (v trenutni različici imamo težave pri daljšem delovanju postaje). Prav tako je na mikrokrmilniku na voljo še 4.324 bajtov (od 32.256 bajtov celotnega pomnilnika) pomnilnika flash, kjer se hrani program.

Na področju amaterskega merjenja vremenskih podatkov nam je na voljo veliko komercialnih rešitev. Naša rešitev bi lahko bila neka osnova za kasnejše projekte, oziroma za kasnejše različice vremenske postaje. Programsko ter fizično rešitev bi lahko tudi skupaj ali posamično prodajali. V tem primeru bi morali razviti rešitev, osnovano na tiskanem vezju, kar bi še zmanjšalo stroške (senzorji ter ostale komponente, namenjene za spajkanje na tiskano vezje, so cenejše) ter povečalo zanesljivost električnih povezav (tiskano vezje je bolj zanesljivo kot povezava senzorjev z žičkami). Prav tako bi lahko v primeru kasnejše uporabe mikrokrmilnika Arduino izdelali lasten ščit, kar bi še poenostavilo priklop senzorjev, omogočalo pa

bi tudi lažjo zasnovo morebitnih razširitev.

Rešitev bi lahko uporabili tudi v sistemih za krmiljenje pametnih hiš, sistemov HVAC (angl. Heating, Ventilation and Air Conditioning systems) ali kot osnovo pri izdelavi raznih senzorskih mrež.

Slike

Tabele

Literatura

[1] "Creative Commons license - Original licenses," http://en.wikipedia.org/wiki/Creative_Commons_license#Original_licenses, 2013, Spletna stran; dostop avgust 2013.

[2] "The GNU General Public License, version 3.0 (GPL-3.0) — Open Source Initiative," http://opensource.org/licenses/GPL-3.0, 2013, Spletna stran; dostop avgust 2013.

[3] "The GNU Lesser General Public License, version 3.0 (LGPL-3.0) — Open Source Initiative," http://opensource.org/licenses/LGPL-3.0, 2013, Spletna stran; dostop avgust 2013.

[4] "Apache License, Version 2.0 — Open Source Initiative," http://opensource.org/licenses/Apache-2.0, 2013, Spletna stran; dostop avgust 2013.

[5] "Apache License 2.0 (Apache-2.0) Explained in Plain English," http://www.tldrlegal.com/license/apache-license-2.0-%28apache-2.0%29, 2013, Spletna stran; dostop avgust 2013.

[6] "The MIT License (MIT) — Open Source Initiative," http://opensource.org/licenses/MIT, 2013, Spletna stran; dostop avgust 2013.

[7] "MIT License Explained in Plain English," http://www.tldrlegal.com/license/mit-license, 2013, Spletna stran; dostop avgust 2013.

[8] "BSD 3-Clause License (Revised) Explained in Plain English," http://www.tldrlegal.com/license/bsd-3-clause-license-%28revised%29, 2013, Spletna stran; dostop avgust 2013.

[9] "GNU General Public License v3 (GPL-3) Explained in Plain English," http://www.tldrlegal.com/license/gnu-general-public-license-v3-%28gpl-3%29, 2013, Spletna stran; dostop avgust 2013.

[10] "LGPL3 License Explained in Plain English," http://www.tldrlegal.com/license/gnu-lesser-general-public-license-v3-%28lgpl-3.0%29, 2013, Spletna stran; dostop avgust 2013.

[11] "Mozilla Public License 2.0 (MPL-2) Explained in Plain English," http://www.tldrlegal.com/license/mozilla-public-license-2.0-%28mpl-2%29, 2013, Spletna stran; dostop avgust 2013.

[12] "Arduino - ArduinoBoardEthernet," http://arduino.cc/en/Main/ArduinoBoardEthernet, 2013, Spletna stran; dostop maj 2013.

[13] P. Unterdorfer, "Power over Ethernet - IEEE 802.3af," Hirschmann Automation and Control GmbH, Tech. Rep., junij, Dostopno na http://belden.com/docs/upload/PoE_Basics_WP.pdf; Dostop Maj 2013.

[14] "Arduino - Wire," http://arduino.cc/en/reference/wire, 2013, Spletna stran; dostop avgust 2013.

[15] "Humidity Sensor SHT75: Sensirion AG," http://www.sensirion.com/en/products/humidity-temperature/humidity-sensor-sht75/, 2013, Spletna stran; dostop avgust 2013.

[16] "Datasheet SHT7x (SHT71, SHT75), Humidity and Temperature sensor IC. Sensirion, the Sensor Company." http://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/Humidity/Sensirion_Humidity_SHT7x_Datasheet_V5.pdf, 2013, splet; dostop maj 2013.

[17] "BMP085 Digital pressure sensor Data sheet - Bosch Sensortec," http://www.adafruit.com/datasheets/BMP085_DataSheet_Rev.1.0_01July2008.pdf, 2013, splet; dostop maj 2013.

[18] "Welcome to KDevelop.org — KDevelop," http://kdevelop.org/, 2013, Spletna stran; dostop avgust 2013.

[19] "Git," http://git-scm.com/, 2013, Spletna stran; dostop avgust 2013.

[20] "Git - About," http://git-scm.com/about/small-and-fast, 2013, spletna stran; dostop maj 2013.

[21] "GitHub · Build software better, together." https://github.com/, 2013, Spletna stran; dostop avgust 2013.

[22] "About the Apache HTTP Server Project - The Apache HTTP Server Project," http://httpd.apache.org/ABOUT_APACHE.html, 2013, spletna stran; dostop maj 2013.

[23] "PHP: Copyright," http://php.net/copyright.php, 2013, Spletna stran; dostop avgust 2013.

[24] "Usage Statistics and Market Share of Server-side Programming Languages for Websites, May 2013," http://w3techs.com/technologies/overview/programming_language/all, 2013, spletna stran; dostop maj 2013.

[25] "PHP: PHP Usage Stats," http://www.php.net/usage.php, 2013, spletna stran; dostop maj 2013.

[26] "PHP: PostgreSQL - Manual," http://php.net/manual/en/book.pgsql.php, 2013, spletna stran; dostop maj 2013.

[27] "PHP: ODBC - Manual," http://php.net/manual/en/book.uodbc.php, 2013, spletna stran; dostop maj 2013.

[28] "Highcharts - Interactive JavaScript charts for your webpage," http://www.highcharts.com/, 2013, Spletna stran; dostop avgust 2013.

[29] "ATAVRSBPR1 - ATMEL - BMP085, PRESSURE SENSOR, EVAL — Farnell Republika Slovenija," http://si.farnell.com/atmel/atavrsbpr1/bmp085-pressure-sensor-eval-board/dp/1972206, 2013, Spletna stran; dostop junij 2013.

[30] "Kate — Get an Edge in Editing — The Kate Editor Homepage," http://kate-editor.org/, 2013, Spletna stran; dostop avgust 2013.

[31] "Arduino Playground - sensirion," http://playground.arduino.cc/code/Sensirion, 2013, Spletna stran; dostop junij 2013.

[32] "Pareto's Principle - The 80-20 Rule," http://management.about.com/cs/generalmanagement/a/Pareto081202.htm, 2013, spletna stran; dostop junij 2013.

[33] Wikipedia, "Avalanche effect - Wikipedia, The Free Encyclopedia," http://en.wikipedia.org/w/index.php?title=Avalanche_effect&oldid=540522053, 2013, Spletna stran; dostop junij 2013.