Eetu Purtonen

Kodin kulunvalvontajärjestelmä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Ohjelmistotekniikka

Insinöörityö

22.11.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Eetu Purtonen Kodin kulunvalvontajärjestelmä 29 sivua + 1 liite 22.11.2018
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tietotekniikka
Ammatillinen pääaine	Ohjelmistotekniikka
Ohjaaja	Lehtori Ilpo Kuivanen

Työn tarkoituksena oli tuottaa avoimista laitteista ja lähdekoodista muodostuva yksinkertainen kodin kulunvalvontajärjestelmä, joka koostuu liiketunnistimesta, kamerasta ja sähköisesti ohjattavasta magneettilukosta. Konkreettisen ohjelmiston lisäksi tavoitteena oli laiteohjelmointiin syventyminen.

Laitteiston perustana toimii Raspberry Pi -tietokone, jonka GPIO-pinneihin kytkettiin ohjattavat ja luettavat laitteet.

Ohjelmisto koostuu kolmesta erillisestä osasta, jotka keskustelevat keskenään web-sokettien ja Unix-sokettien avulla. Ohjelmiston osia ovat C++-kielellä kirjoitettu laiteohjain, Node.js:llä kirjoitettu verkko-ohjain ja selaimessa toimiva Javascript-kieltä ja jQuery-kirjastoa hyödyntävä käyttöliittymä.

Työn lopputuloksena saatiin toimiva yksinkertainen kulunvalvontajärjestelmä ja paljon uutta tietoa ja kokemusta laiteohjelmoinnista ja prosessien välisestä kommunikoinnista, mikä oli alkuperäinen tarkoitus.

Työn raportissa kerrotaan käytetystä laitteistosta, ohjelmiston eri osien ohjelmointikielien valinnasta, ohjelmiston infrastruktuurista ja käytetyistä kehitystyökaluista ja -ympäristöistä.

Avainsanat	Kulunvalvonta, Raspberry Pi, C/C++, node.js, jQuery, Unix socket, websocket

Author Title Number of Pages Date	Eetu Purtonen Home access control system 29 pages + 1 appendix 22 November 2018
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Professional Major	Software engineering
Instructors	Ilpo Kuivanen, Senior Lecturer

The goal of this project was to create a simple, open source home access control system consisting of a motion sensor, a camera and an electronically controllable magnetic lock. In addition to the concrete software system there was also a personal goal to learn more about embedded systems and how to program them.

As the base of this project, a Raspberry Pi single board computer was used to which's GPIO pins all the devices were connected.

The software consists of three distinct parts that communicate using Unix domain sockets and websockets. These parts are a device controller written in C++, a web controller written in Node.js and a user interface running in the browser, powered by Javascript and jQuery.

As the end result a working simple access control system was made as well as a lot of knowledge and experience on embedded programming and communications between processes written in different languages, those which were the beginning goal of this project.

The report is divided into sections that talk about the selected hardware, selecting programming languages for different parts of the system, infrastructure of the software as well as tools and development environment used in the development of this project.

Keywords	Access control, Raspberry Pi, C/C++, node.js, jQuery, Unix socket, websocket

Sisällys

Lyhenteet

IPC	Inter-Process Communication. Eri ohjelmien tai prosessien välinen kommunikaatio, esimerkiksi verkon yli tai järjestelmän sisäisiä putkia pitkin.
GPIO	General Purpose Input/Output. Elektroniikassa käytetty yleiskäyttöinen pinni, joka voidaan ohjelmoida joko lähettämään tai vastaanottamaan signaalia.
IOT	Internet Of Things. Termi, jota käytetään yleisesti kuvaamaan erilaisten elektronisten laitteiden hallintaa ja datan käsittelyä internetin yli. Esimerkiksi puhelimella internetin yli hallittava ilmastointi.
GCC	GNU Compiler Collection. GNU-projektin eri kielien kääntäjien kokoelma. Mahdollistaa mm. C-, C++- ja Go-kielisten koodien kääntämisen.
NPM	Node Package Manager. Javascript-ohjelmistojen pakettien hallinta, jolla voidaan ylläpitää ohjelmiston riippuvuuksia ja asentaa kehitystyökaluja.
SSH	Secure Shell. Kryptografinen verkkoprotokolla, jonka avulla voi turvallisesti etäohjata laitteita komentorivillä turvattoman tai julkisen verkon yli.
SFTP	SSH File Transfer Protocol tai Secure File Transfer Protocol. Verkkoprotokolla, joka mahdollistaa tiedostojen lukemisen, muokkauksen ja siirron hyödyntäen SSH-yhteyttä.
SSHFS	SSH File System. SSH-yhteydellä verkon yli toimiva tiedostojärjestelmä, joka mahdollistaa toisella verkossa olevalla laitteella olevien tiedostojen ja kansioiden käytön, kuin ne olisivat käyttäjän omalla laitteella.
GPAC	Avoimen lähdekoodin multimediaohjelmistokehys, joka sisältää useita eri multimediatyökaluja.
MP4Box	GPACin sisältämä multimediatyökalu videotiedostojen paketoimiseen MP4-muotoon.

1 Johdanto

Työn tavoitteena oli ensisijaisesti oman osaamisen laajentaminen laitteisto-ohjelmoinnin ja suurempien ohjelmistokokonaisuuksien suunnittelun ja toteutuksen osalta. Lopputuloksena ei siis ole täydellinen valmis tuote, vaan enemmänkin prototyyppi ja soveltuvuusselvitys (”proof of concept”). Vaikka tein työn vain itseäni varten ja oppimistarkoituksessa, yritin parhaani mukaan asettaa vaatimuksia ohjelmistolle ja valita ominaisuuksia, joita vastaavissa tosielämän kaupallisissa järjestelmissä on.

Vastaavat kaupalliset järjestelmät ovat myös ohjelmiston sekä laitteiston puolesta suljettuja, niiden käyttö on mahdollista vain kuukausi- tai vuosimaksullisten lisenssien avulla. Tämä tarkoittaa myös sitä, että ongelmatilanteissa täytyy aina tilata toimittajalta huoltotyö. Lisäksi laitteisto ja ohjelmisto ovat tiukasti sidottuna toisiinsa, eli molemmat täytyy aina ostaa samalta toimittajalta, eikä esimerkiksi olemassa olevan laitteiston käyttäminen toisen toimittajan ohjelmistolla ole mahdollista. Tässä työssä kaikki laitteet ovat eri valmistajilta ja niissä on hyvin vähän rajoittavia tekijöitä yhteensopivuuden kannalta. Käytännössä siis jokaisen laitteiston osan vaihtaminen toiseen onnistuu minimaalisilla muutoksilla tai ilman muutoksia ohjelmistoon.

Vertailukohteena käytän Stanley Security Oy:n kaupallista Timecon®-tuoteperhettä, koska minulla on sen käytöstä ja huollosta henkilökohtaista kokemusta aiemman työsuhteeni kautta. Timecon®-tuoteperheeseen kuuluu kulunvalvonnan, työajanseurannan ja rikosilmoittimien laitteistot ja ohjelmistot. Stanley Securityllä on myös kameravalvontaan omat laitteistot ja ohjelmistot, mutta niistä minulla ei ole omakohtaista kokemusta.

Kulunvalvonta

Kulunvalvontaan kuuluu valvotun tilan ovien tai porttien elektroninen ohjaus ja tilaseuranta. Ensimmäinen vaatimus järjestelmälle oli siis elektronisesti ohjattava lukko, jota voi ohjata sekä etänä tietokoneelta, että oven välittömästä läheisyydestä löytyvällä lukijalla. Lukijan tuli toimia henkilökohtaisilla korteilla, joille määriteltiin kulkuoikeudet oveen.

Tavallisesti muut valvontalaitteet kuten kamerat, liiketunnistimet ja lasirikkoilmaisimet kuuluvat omiin erillisiin järjestelmiinsä, mutta tässä työssä samassa järjestelmässä tuli olla myös liiketunnistin sekä kamera. Liiketunnistimen havaitessa liikettä, aktivoidaan kamera, joka ottaa kuvan tai lyhyen videon kohteesta ja tallentaa sen.

Ohjelmiston käyttöliittymä

Järjestelmän hallintaan luotiin käyttöliittymä, jonka kautta voi reaaliajassa ohjata lukkoa, seurata liiketunnistimen tilaa, sekä tarkastella kameran tallentamia kuvia tai videoita. Reaaliaikaisen seurannan lisäksi käyttöliittymästä tuli nähdä tapahtumaloki, josta ilmenee liiketunnistimen hälytykset ja lukijan tapahtumat, joihin kuuluu

- luetun kortin numero
- kortin haltijan nimi
- onko kortti hyväksytty eli onko lukko avattu.

Käyttöliittymään pääsyä tuli rajoittaa niin, että kuka tahansa ulkopuolinen ei pääse järjestelmään ja avaamaan ovea itselleen ilman kulkuoikeutta.

2 Laitteisto ja kytkennät

2.1 Raspberry Pi

Työn perustana toimii Raspberry Pi -tietokone sen pienen koon, virrankulutuksen ja helposti ohjelmoitavien GPIO-pinnien takia.

Raspberry Pi on yhden piirilevyn tietokone, eli kaikki tietokoneen komponentit (prosessori, muisti, näytönohjain yms.) ovat juotettuna yhdelle piirilevylle, toisin kuin perinteisissä tietokoneissa, joissa komponentit liitetään erilaisia kantoja ja liittimiä käyttämällä emolevylle ja ovat helposti vaihdettavissa. [1.]

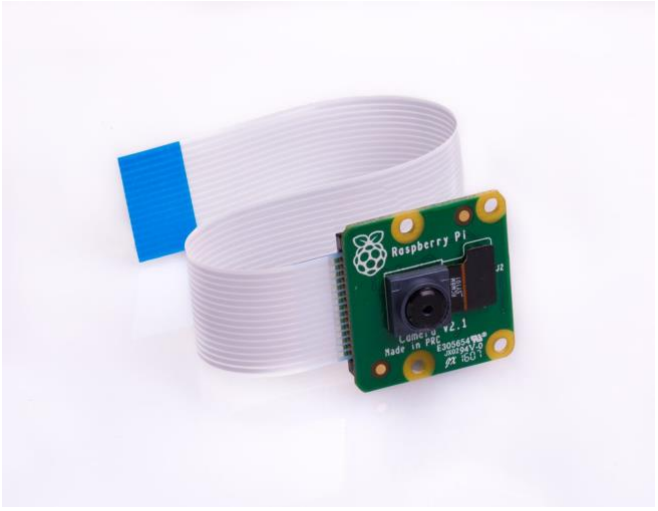
Kuva 1. Raspberry Pi 3 model B

Nimessä mainittu model B viittaa piirin ulkoasuun, liittimien määrään ja sijoitteluun. Mallimerkintöjä on tällä hetkellä A, A+, B, B+, sekä pienemmässä Raspberry Pi Zero -piirilevyssä mallit W ja WH. Merkittävimmät erot A- ja B-mallien välillä on piirin fyysinen koko ja A-mallissa ethernet-portin puuttuminen. W- ja WH-mallien ainoa ero on piirin GPIO-pinneissä. W-mallissa GPIO-pinnejä itsessään ei ole, vaan piirilevyllä on pelkät reiät, joihin käyttäjän tulee juottaa kiinni tarvittavat komponentit ja/tai liitinjohdot. [2.]

Ohjelmoitavat GPIO-pinnit voidaan asettaa joko sisään- tai ulostuloksi, ne toimivat 3,3 voltin jännitteellä siten, että ulostulona ohjataan jännite päälle ja pois tai sisääntulona pinniin asetetaan 3,3 voltin jännite ja Raspberry Pi mittaa jännite-eron pinnin ja maan välillä. Tietyn rajan ylittävä jännitehäviö tulkitaan sisääntulon piirin avautumiseksi ja sisääntulon arvo on tällöin ”Low” eli 0. [3.]

2.2 Raspberry Pi -Kameramoduuli

Vaikka kaikki oleelliset komponentit ovat juotettuna yhdelle piirilevylle, on Raspberry Pi:hin mahdollista liittää erilaisia oheislaitteita kuten kameramoduuli, joka kytketään sille varattuun omaan liittimeen piirilevyllä. Raspberry Pi 3 model B piirilevylle on myös mahdollista kytkeä pieni nestekidenäyttö, mutta tässä työssä sille ei ole tarvetta.



Kuva 2. Raspberry Pi -Kameramoduuli V2

Koska moduuli on Raspberry Pi:n valmistama ensimmäisen osapuolen tuote, sen käyttö on automaattisesti tuettu virallisessa Raspbian käyttöjärjestelmässä ilman erillisten kirjastojen tai ohjelmien asennusta. [4.]

2.3 Infrapuna liiketunnistin

Liikkeen tunnistamiseen käytin työssä turva-alalla erittäin yleistä infrapunasensoria. Sensori mittaa ympäristön tuottaman infrapunasäteilyn määrää ja säteilyn määrän muuttuessa laukaisee piirin ulostulokytkimen. Toisin kuin etäisyyttä ja nopeutta mittaavat ultraäänellä toimivat tutkat, jotka lähettävät ultraääniä ja mittaavat takaisin heijastuvan signaalin. Infrapunasensori ei lähetä infrapunasäteilyä vaan mittaa ympäristön luonnollisesti tuottamaa säteilyä.

Kuva 3. Securitaksen liiketunnistin. Kuvassa väärin päin, laite asennetaan kattotasoon, jolloin harmaa sensoriosa osoittaa viistoon alaspäin.

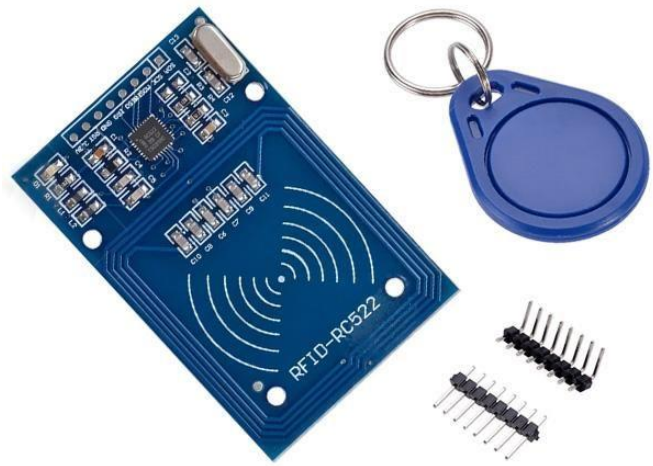
Sensorin piirissä on kaksi ulostuloa, NC (Normally Closed) ja NO (Normally Open), sekä COM (Common), jota vasten ulostulo mitataan. NC tarkoittaa, että normaalitilanteessa ulostulo ja COM ovat yhdistettynä, eli piiri on suljettu ja virta kulkee piirin lävitse. Kun sensori havaitsee liikkeen, ulostulo katkaistaan eikä piirissäi kulje enää virta. NO taas toimii päinvastoin, eli normaalitilanteessa piiri on avoin eikä siinä kulje virta, liikkeen havaittuaan sensori sulkee piirin ja virta kulkee piirin lävitse. Suurin osa liiketunnistimista kytketään käyttäen NC-ulostuloa, eli piirissä kulkee virta ja virran katkeaminen indikoi liikettä. Tämä tehdään siksi, ettei sensoria saisi ”huijattua” katkaisemalla signaalia kuljettavaa johdinta, sillä se laskettaisiin liikkeen tunnistamiseksi.

2.4 MFRC522 RFID -lukija

RFID, eli ”Radio Frequency Identification”, on menetelmä, jolla voidaan lukea RFID-tunnisteita lyhyen matkan päästä. Toimintaperiaate on sama kuin moderneista maksukorteista löytyvä lähimaksu. Kortissa on pieni mikrosiru ja siihen yhteydessä oleva antenni. Antenni indusoi lukijan lähettämästä radiotaajuisesta signaalista erittäin pienen käyttövirran, piiri lähettää lukijalle takaisin korttiin määritellyn ID-numeron. [5.]

Jotta ihmisten kulkua voidaan valvoa ja hallita, tarvitaan kulkijoille jonkinlainen tunniste ja tämän tunnisteen lukemiseen lukija. Valitsin tähän Raspberry Pin kilpailevalle valmistajalle, Arduinolle, tarkoitetun MFRC522 RFID -lukijan. Kyseisen lukijan valinta

perustui vain yksinkertaisesti siihen, että lukija toimii Raspberry Pin tuottamalla 3,3 V:n jännitteellä ja lukijan käyttöön löytyi valmis C++-kirjasto, joka on virallisesta Arduino-kirjastosta Raspberry Pille haarautettu versio. Ebaysta tilatessa pakettiin kuului myös yhteensopiva tunnistekortti ja -tägi.



Kuva 4. MFRC522 RFID-lukija

2.5 Abloy Assa magneettilukko

Jotta ihmisten kulkua voidaan valvoa ja rajoittaa, tarvitaan myös elektronisesti ohjattava lukko oveen. Työn luonteen kannalta ei ollut järkevää alkaa asentamaan oikean oven sisään tulevaa lukkorunkoa, joten valitsin käyttötarkoitukseen sopivamman magneettilukon, jonka voi asentaa käytännössä mihin tahansa, sillä se toimii pinta-asennuksena sen sijaan, että lukko täytyisi työntää oven sisään.

Magneettilukko poikkeaa tavallisista lukoista siinä, ettei siinä ole salpaa tai mitään muutakaan liikkuvaa osaa, jolla ovi lukittaisiin. Sen sijaan oven lukitseminen tapahtuu asentamalla magneettilukko seinään, kattoon tai lattiaan ja lukon vastakappale oven reunaan. Lukon ja vastakappaleen ollessa kosketuksissa tai lähellä toisiaan, voidaan ovi lukita syöttämällä lukkoon virtaa, joka aktivoi sähkömagneetin ja vetää ovessa olevaa vastakappaletta puoleensa, jolloin ovea ei voi vetää auki.

Magneettilukkoja käytetään usein lasi- tai liukuovissa, joihin tavallista lukkoa ei saa asennettua tai paikoissa, joissa tiloista ulos pääsy halutaan estää. Tässä työssä magneettilukkoa käytettiin lähinnä sen saatavuuden ja asennuksen helppouden takia.

Kuva 5. Magneettilukko ja vastakappale. Kuvan lukko eroaa työssä käytetystä lukosta.

2.6 Virtalähde

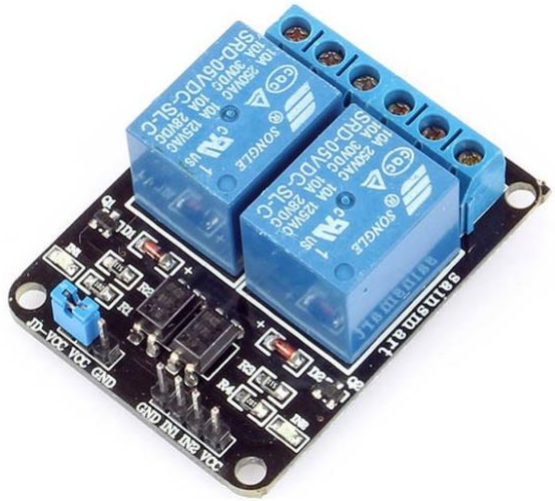
Magneettilukko ja liiketunnistin vaativat toimiakseen 12 tai 24 voltin tasavirran, joten työhön täytyi lisätä vielä erillinen virtalähde. Virtalähde kytketään verkkovirtaan tavalliseen sisätilapistokkeeseen ja ulostuloon pikakiinnityksellä johtimet.

Kuva 6. Mascot 12/24V virtalähde. Kuvan virtalähde eroaa hiukan työssä käytetystä virtalähteestä.

2.7 Relekortti

Magneettilukko toimii 12 tai 24 voltin jännitteellä, jota Raspberry Pi ei pysty tuottamaan eikä kestäisi piirissä niin suurta jännitettä, joten lukon ohjaamiseen tarvittiin relettä, jolla voidaan ohjata magneettilukkoon syötettävää virtaa päälle ja pois Raspberry Pin GPIO-pinnien 3.3 voltilla.

Valittu relekortti on myös alun perin Arduinolle valmistettu lisäkortti, mutta toimii myös Raspberry Pillä. Kortilla on kaksi erikseen ohjattavaa relettä, jotka voidaan ohjata auki tai kiinni syöttämällä valitun releen sisääntuloon 3.3 voltin jännite. Ohjausjännitteen lisäksi kortti vaatii 5 voltin toimintajännitteen, joka saadaan Raspberry Pin 5V GPIO-pinnistä.

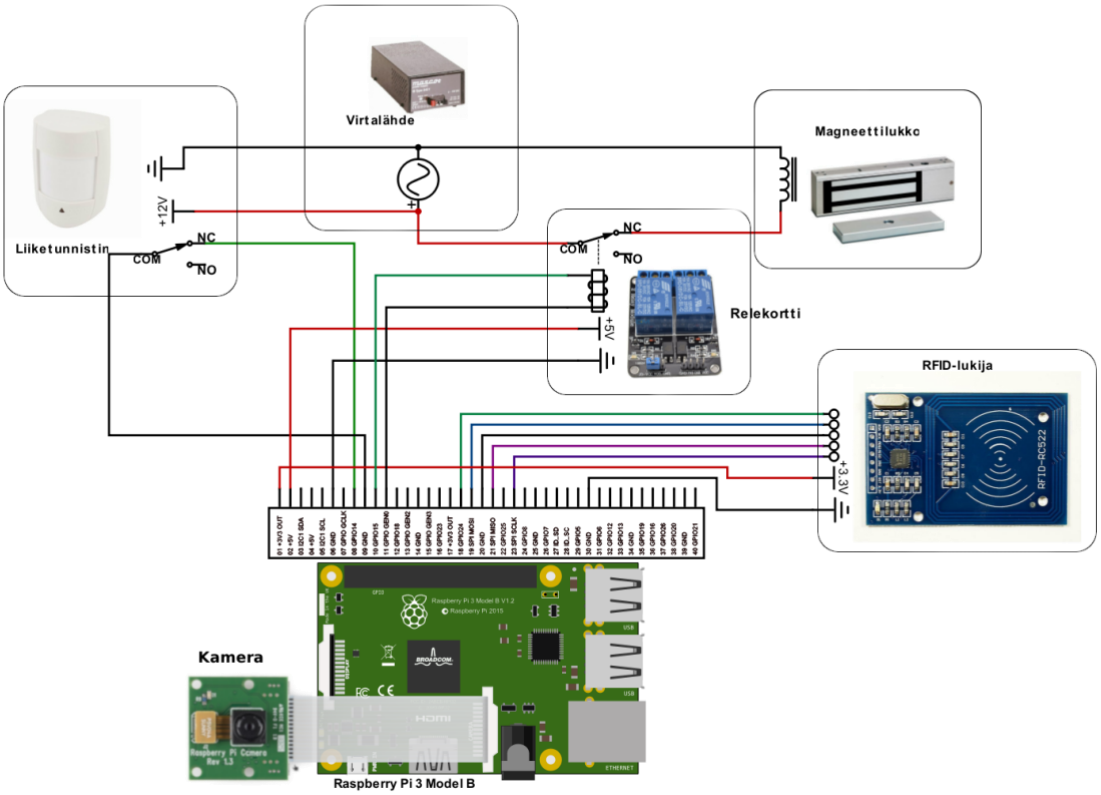


Kuva 7. Kaksikanavainen relekortti

Releiden ulostulossa on samat liitännät kuin aiemmin mainitussa liiketunnistimessa, eli NC, NO ja COM. Tässä tapauksessa valitsin lukon kytkettäväksi NC-liittimeen, jolloin normaalitilanteessa piiri on suljettu ja virta kulkee lukolle ja vetää lukon kiinni. Lukon avaaminen tapahtuu syöttämällä releen sisääntuloon 3,3 voltin jännite, jolloin rele vetää ja sen ulostulon piiri aukeaa, katkaisten virran lukolle. [6.]

2.8 Kytkennät

Kuvassa 8 kaikki aiemmin mainitut komponentit on kytketty Raspberry Pin GPIO-pinneihin. Kuvasta puuttuu reitittimeen kytkettävä verkkokaapeli ja Raspberry Pin sekä virtalähteen verkkovirtakaapelit. [7.]



Kuva 8. Järjestelmän kytkennät

Käytin työssä mahdollisimman avoimia ja yksinkertaisia laitteita, jotta järjestelmän muokattavuus ja kehitys olisi mahdollisimman helppoa. Vertailukohteena Timeconissa järjestelmä koostuu täysin suljetuista laitteista ja lähdekoodista ja järjestelmässä on mahdollista käyttää vain yhteensopivia laitteita. Timeconissa suuremmat järjestelmät käyttävät useampia fyysisiä ohjainkortteja, joihin voidaan kytkeä pienempi määrä sensoreita, lukijoita ja lukkoja, ja ohjainkortit keskustelevat yhden pääohjaimen kanssa, jota ohjelmistolla sitten hallitaan. Ohjainkortit ovat suuria, jopa pienen kannettavan tietokoneen kokoisia, joten ne vaativat lähes poikkeuksetta oman tilan tai huoneen, johon kortit voidaan asentaa. Tässä työssä käytetty laitteisto ja ohjelmisto mahdollistaisi ohjaimen vaihdon mihin tahansa pieneen piirilevyyn, jossa on ohjelmoitavia GPIO-pinnejä, verkkokortti ja mahdollisuus ajaa laitekoodia. Esimerkiksi erittäin minimaalinen

4 cm x 4 cm FriendlyARM NanoPi Neo. FriendlyARM NanoPi Neon voisi helposti piilottaa, vaikka oven sisään. Kameraliitäntää siinä ei ole, mutta USB-porttiin voisi liittää tavallisen webkameran, joka tekisi järjestelmästä vielä enemmän muokattavan.

Kuva 9. FriendlyARM NanoPi Neo

3 Kielen ja ohjelmistokehysten valinta

Varsinainen ohjelmointiosuus projektista alkoi valitsemalla projektiin sopiva ohjelmointikieli. Tässä luvussa kuvataan muutaman eri kielen eroja, sekä vertaillaan niiden soveltuvuutta projektiin.

3.1 Python

Ensimmäinen looginen vaihtoehto kielen valintaan oli Python. Pythonilla on suurin tuki ja eniten valmiita koodipohjia Raspberry Pi:n ohjelmointiin, sillä se on Raspberry Pi:n virallinen tuettu kieli ja Raspberry Pi oli alun perin tehty Pythonin opettamista varten. [8.]

Pythonin hyviä puolia on sen syntaksin yksinkertaisuus ja kyky suorittaa ohjelmia ilman erillistä kääntäjää. Syntaksi on tarkoitettu helposti luettavaksi ja muista ohjelmointikielistä tutut ohjelmablokkeja ympäröivät aaltosulkeet ja komentoja päättävät puolipisteet eivät ole pakollisia. Lisäksi syntaksissa käytetään paljon englanninkielisiä avainsanoja erikoismerkkien sijaan. Esimerkkikoodissa 1 on yksinkertainen konditionaalilause Pythonilla ja C:llä.

```
do_this() if condition1 or condition2 else do_that()
```

Esimerkkikoodi 1. Pythonilla kirjoitettu yhden rivin helppolukuinen konditionaalilause, jossa suoritetaan funktio ”do_this”, jos jompikumpi ”condition1” tai ”condition2” toteutuu. Muussa tapauksessa suoritetaan funktio ”do_that”.

```
if(condition1 || condition2){
     do_this();
} else {
     do_that();
}
```

Esimerkkikoodi 2. Sama koodi kuin esimerkissä 1, mutta kirjoitettuna pidemmässä muodossa perinteisen ohjelmointisyntaksin mukaan, kuten esimerkiksi C/C++, Java, tai C#.

Python on tulkattava ohjelmointikieli, eli komennot suoritetaan yksi kerrallaan ajon aikana kääntämällä ne ensin konekielisiksi komennoiksi ja sitten suorittamalla niitä. Tämä helpottaa ja nopeuttaa koodin ajonaikaista testaamista ja vianetsintää, kun taas käännettävissä kielissä koodin kääntäminen voi kestää jopa useita minuutteja.

3.2 Node.js

Node.js on palvelimella ajettava versio normaalisti selaimessa ajettavasta Javascriptistä. Nodea kutsutaan usein ohjelmointikieleksi, vaikka se ei sitä varsinaisesti kuitenkaan ole, vaan ajonaikainen ympäristö, joka kääntää Javascript-koodin ajon aikana konekielelle, jota palvelin sitten suorittaa. Kääntämien tapahtuu Googlen V8-moottorilla, joka on Google Chromen käyttämä ajonaikainen kääntäjä Javascript koodille. Node on julkaistu 2009, joten se on erittäin uusi, joka tekee siitä moderniin sivustokehittämiseen erittäin hyvän ja varteenotettavan vaihtoehdon. [9.]

Noden vahvuuksiin kuuluu muun muassa web-sokettien käyttö, joka mahdollistaa kaksisuuntaisen yhteyden selaimen ja palvelimen välillä. Perinteisessä yksisuuntaisessa kommunikaatiossa selain eksplisiittisesti pyytää palvelimelta jotakin resurssia (HTML-dokumentti, tyylitiedostot, kuvat yms.), jonka palvelin palauttaa ja tämän jälkeen yhteys suljetaan, eikä kummassakaan päässä tiedetä toisesta ennen seuraavaa selaimesta tehtävää pyyntöä. Web-soketeilla mahdollistetaan avatun yhteyden pitäminen auki, jolloin myös palvelimen päästä voidaan lähettää selaimelle dataa ja pyyntöjä ilman, että selaimen päästä tehdään kyselyä palvelimelle. Tässä työssä web-sokettien käyttö on tärkeää, koska ohjelman käyttöliittymä on vain selaimessa ja palvelimen muutokset tulisi muuttaa käyttöliittymää reaaliajassa.

3.3 C/C++

Vaikka C ja C++ ovat omia kieliään, nimeä C/C++ käytetään usein kuvaamaan C++-ohjelmia, koska C++:n syntaksi mahdollistaa myös C-kielisten komentojen ja kirjastojen käytön. C++ on siis luotu alkuperäisen C:n pohjalta, johon on lisätty ohjelmoijan työnkulkua helpottamaan ominaisuuksia kuten luokat ja oliot.

C- ja C++-kielillä kirjoitetut ohjelmat ovat erittäin suorituskykyisiä ja kevyitä. Parempi suorituskyky johtuu siitä, että C/C++ koodi käännetään ennen suorittamista konekielelle, jota prosessori pystyy suoraan ajamaan sen sijaan, että suorituksen aikana jokainen suoritettava käsky tulkitaan konekielen käskyksi, jonka prosessori sitten suorittaa. Käännösvaiheessa koodi myös tarkistetaan virheiden varalta, jolloin ajon aikaisia virheitä on mahdollista syntyä huomattavasti vähemmän.

Huono puoli C- ja C++-kielissä on niiden monimutkaisuus ja vaikeus. Monet asiat kuten roskien kerääminen (käyttämättömien muistipaikkojen vapauttaminen) hoituu muissa kielissä automaattisesti, jolloin ohjelmoijan tarvitse välittää siitä koodia kirjoittaessa. Manuaalinen muistinhallinta taas on lähes pakollista ohjelmoitaessa esimerkiksi sulautettuja järjestelmiä, joissa muistin määrä on erittäin rajallinen. [10.]

3.4 PHP

PHP on yksi suosituimpia verkko-ohjelmointikieliä sen yksinkertaisuuden takia ja soveltuu tästä syystä hyvin nopeiden verkkosovellusten tai prototyyppien tekemiseen. PHP on Pythonin tapaan tulkattava ohjelmointikieli, eli koodia ei käännettä laitekoodiksi, kuten esimerkiksi C:tä ja C++:aa vaan evaluoidaan ajon aikana. PHP, kuten myös Javascript, on dynaamisesti tyypitetty, eli muuttujien tyyppiä ei tarvitse erikseen määritellä ja eri tyyppisten muuttujien vertailu keskenään onnistuu ilman tyyppimuunnoksia. [11.]

Web-sokettien ohjelmointi PHP:lla onnistuu kyllä, mutta vaatii huomattavasti enemmän koodia kuin esimerkiksi nodella. PHP ei myöskään ole laiteohjelmointiin tarvittavan matalan tason kieli.

3.5 Lopputulos

Ensimmäisenä kielenä kokeilin Pythonia sen helppouden takia. Koska käyttöliittymä on selaimessa, kokeilin myös Pythonilla kehitettyä verkko-ohjelmointiin tarkoitettua ohjelmointikehystä, Djangoa. Django kuitenkin osoittautui käyttötarkoitukseen turhan raskaaksi, eikä selaimen ja palvelimen välinen kaksisuuntainen kommunikaatio web-soketeilla vaikuttanut olevan kovin helppoa. [12.]

Pythonille löytyi myös toinen kevyempi ohjelmistokehys, Tornado, joka on tarkoitettu nimenomaan kaksisuuntaiseen kommunikaatioon. Jonkin aikaa kehykseen tutustuttuani kuitenkin huomasin dokumentaation olevan puutteellista ja vaikealukuista, eikä tarkoitukseen sopivia esimerkkikoodeja löytynyt.

Tämän jälkeen päätin hylätä Pythonin käytön ja vaihtaa lähestymistapaa. Sen sijaan että valitsisin kielen, joka soveltuu tyydyttävästi kaikkeen, valitsin eri tehtäviin eri kieliä. C/C++ soveltuu laitteiston ohjelmointiin, joten käytin sitä GPIO-pinnien lukemiseen ja ohjaamiseen. C/C++ ei taas sovellu verkko-ohjelmointiin, joten siihen käytin Nodea, jolla web-sokettien käyttö on helppoa ja nopeaa. PHP:n käyttö ei ollut varsinaisesti tarvittavaa, mutta sillä on helppo jakaa yksi iso HTML-dokumentti useisiin pienempiin osiin koodin luettavuuden helpottamiseksi. Ohjelmistojen eri osien välillä täytyi vielä saada jokin yhteys, jotta kommunikaatio eri kielisten prosessien välillä onnistuu.

Verkkopalvelimena toimi kevyt ja suorituskykyinen NGINX ja selaimen pään logiikka toteutettiin perinteisellä Javascriptillä ja jQueryllä. Raskaampien ja ominaisuus-rikkaampien Javascript-ohjelmistokehysten, kuten Angularin tai Reactin, käyttäminen ei tämän työn kannalta ollut tarpeellista, koska suurin osa logiikasta tapahtuu palvelimen päässä. Tietokantaa tarvittiin tässä työssä vain lokin ja kulkuoikeuksien ylläpitämiseen, joten valitsin yksinkertaisen ja kevyen SQLiten.

4 Ohjelmiston infrastruktuuri

Ohjelmistokokonaisuus koostuu 3 erillisestä osasta, jotka kommunikoivat keskenään. Ohjelmiston osia ovat laiteohjain, verkko-ohjain ja käyttöliittymä.

4.1 Prosessien välinen kommunikaatio

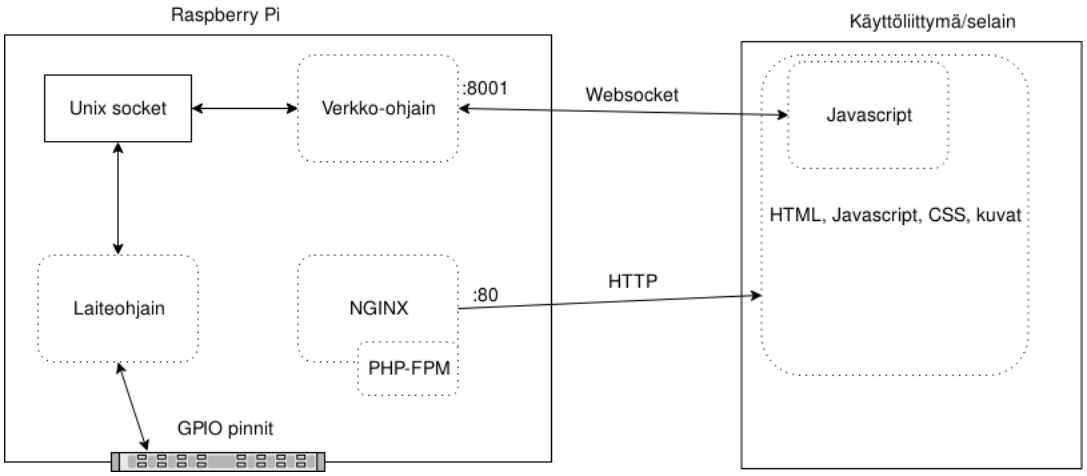
Ohjelmiston jakaminen eri kieliin ja prosesseihin tarkoittaa sitä, että prosessien välille tarvitaan jonkinlainen yhteys, jotta ohjelmiston osat voivat keskustella keskenään.

Usein eri ohjelmistojen kommunikaatio tapahtuu verkon yli jonkinlaisen rajapinnan avulla, koska ohjelmistoja ajetaan eri palvelimilla. Myös saman palvelimen sisällä voidaan hyödyntää TCP-portteja kommunikointiin, mutta tällöin täytyy olla tarkkana, ettei portteja paljasteta verkolle. TCP on tietoliikenneprotokolla, jonka avulla voidaan luoda yhteyksiä verkon yli.

Tämän työn kannalta TCP-porttien käyttäminen ei kuitenkaan ollut välttämätöntä, sillä prosesseja suoritetaan vain saman palvelimen sisällä ja yhteyden kierrättäminen TCP-porttien kautta tuottaa turhaa ylimääräistä suoritusaikaa protokollan ohjatessa lähetettyjä ja vastaanotettuja paketteja. Kommunikaatioon täytyi siis keksiä tehokkaampi protokolla.

Koska Raspberry Pi:llä käytetty Linux-käyttöjärjestelmä on Unix-pohjainen, voidaan kommunikaatioon hyödyntää Unix domain -sokettia, joka on nimenomaan prosessien väliseen kommunikaatioon tarkoitettu. Unix domain -soketti toimii samaan tapaan kuin TCP, mutta tietoliikenne tapahtuu kokonaan käyttöjärjestelmän kernelin, eli ytimen, sisällä ilman verkkoprotokollia. Tämä tekee yhteydestä nopeamman verrattuna TCP-porttien käyttöön. [14.]

Unix domain -soketti käyttäytyy ja toimii järjestelmässä kuin tiedosto, sen kautta kommunikointi tapahtuu kirjoittamalla siihen ja lukemalla siitä. Unix domain -soketeissa voidaan eri protokollia määrittämällä käyttää paketointia, tai kaksisuuntaista paketoimatonta tietovirtaa. Valitsin tähän työhön paketoimattoman tietovirran siitä syystä, että viestien pituus ei välttämättä ole tiedossa, enkä halunnut rajoittaa viestien pituutta, jota paketointi vaatii. Lisäksi kaksisuuntaisuus mahdollistaa sen, että sokettiin voidaan samanaikaisesti sekä kirjoittaa että lukea. Tämän takia viestinnässä tulee ottaa huomioon viestien päättymisen merkkaaminen jollain tavalla. Tässä työssä käytin viestien loppumerkkinä puolipistettä, jotta lyhyen aikavälin sisällä lähetetyt viestit saadaan eroteltua toisistaan. [15.]



Kuva 10. Ohjelmiston infrastruktuuri visualisoituna.

4.2 Laiteohjain

Laiteohjain on se ohjelmiston osa, joka kontrolloi työn fyysisiä osia ja siinä on suurin osa tämän työn varsinaisesta ohjelmalogiikasta. Laiteohjain on myös vastuussa tietokantaan yhdistämisestä, lukemisesta ja kirjoittamisesta sekä prosessien väliseen kommunikointiin tarvittavan Unix-soketin luomisesta ja ylläpidosta.

Ohjelman alustusvaiheessa luetaan tietokannasta rekisteröidyt ovet ja niille määritellyt GPIO-pinnit ja luodaan oviolio, josta oven tilaa voidaan lukea ja ovi avata. Ovelle voidaan määritellä kaksi pinniä, toinen sisään- ja toinen ulostuloksi. Ulostuloksi määritelty pinni ohjaa oven lukkoa ja sisääntulosta voidaan lukea oven tilaa. Lisäksi ovelle määritellään tietokannassa kortit, joilla pääsy on sallittu ja lisätään nämä korttien numerot ovioliolle. Oviolioon liitetään myös lukijaolio, joka lukee kortteja ja kyselee ovioliolta, onko luetulle kortille sallittu pääsy.

Ovien alustuksen jälkeen luodaan Unix domain -soketti ja odotetaan verkko-ohjaimen liittymistä siihen. Kun verkko-ohjain on liittynyt sokettiin onnistuneesti, aloitetaan soketista verkko-ohjaimelta tulevien viestien lukeminen.

```
thread socketlistenerThread(listenToSocket, rc, cl);

void listenToSocket(int rc, int cl) {
    char buf[100];
    string delimiter = ";";

    while (1) {
        // Read data from socket
        while ((rc = read(cl, buf, sizeof (buf))) > 0) {
            string msg(buf, rc);
            size_t pos = 0;
            string token;
            // Split data by delimiter ";" and interpret them separately, in case of
concatenation
            while ((pos = msg.find(delimiter)) != string::npos) {
                token = msg.substr(0, pos);
                msgInterpreter(token);
                msg.erase(0, pos + delimiter.length());
            }

        }

        if (rc == -1) {
            perror("read");
            return;
        } else if (rc == 0) {
            printf("EOF\n");
            close(cl);
        }
    }
}
```

Esimerkkikoodi 3. Säie, joka lukee Unix-sokettia ja välittää saapuneet viestit käsittelyfunktiolle

Esimerkkikoodissa 3 luodaan laiteohjaimessa säie, joka lukee jatkuvasti luotua Unix-sokettia, ja mikäli sokettiin on kirjoitettu dataa, pilkkoo datan puolipisteillä erotelluiksi viesteiksi ja välittää ne käsittelyfunktiolle. Käsittelyfunktio etsii viesteistä ennalta määriteltyjä avainsanoja ja kutsuu sen jälkeen tarvittavia funktioita kuten valitun oven avaamiseen tai kameralla kuvan ottamiseen ja tallentamiseen. [16.]

Soketin lukijasäikeen lisäksi käynnistetään myös toinen säie, joka lukee sensoreille rekisteröityjen GPIO-pinnien tilaa ja pitää kirjaa niiden tilasta. Mikäli pinnin tila muuttuu, eli sensori on laukaistu, kirjoitetaan sokettiin JSON-muotoinen lokitapahtuma, jonka verkko-ohjain lukee ja välittää edelleen käyttöliittymälle, joka käsittelee ja visualisoi tapahtuman käyttäjälle. Lisäksi lokitapahtuma kirjataan tietokantaan.

Kolmas ja viimeinen säie on lukijan korttien lukemiseen ja ovien avaamiseen sallitun kortin löytyessä.

4.2.1 RFID-lukijan ohjaus

Lukijalla korttien lukemiseen käytin MFRC522-kirjastoa, joka taas käyttää bcm2835-kirjastoa lukijan ohjaamiseen. Kirjasto tarjoaa metodin kortin tunnistamiseen, jonka jälkeen kortin tunniste voidaan lukea tavutaulukoksi, josta se taas voidaan kääntää merkkijonoksi. Kun kortti on löytynyt ja luettu merkkijonoksi, merkkijono lähetetään lukijaan liitetylle ovioliolle, joka tarkistaa, onko kyseinen kortin numero sallittujen korttien listalla, ja avaa lukon, jos on.

```
MFRC522 mfrc;
mfrc.PCD_Init();
string card;

while (1) {
    // Look for a card
    if (!mfrc.PICC_IsNewCardPresent())
        continue;

    if (!mfrc.PICC_ReadCardSerial())
        continue;

    string card(reinterpret_cast<char const*>(mfrc.uid.uidByte), mfrc.uid.size);
    this->door->cardReadEvent(card);

    usleep(500000);
}
```

Esimerkkikoodi 4. Kortin tunnistus ja lukeminen. [17]

4.2.2 GPIO-pinnien ohjaus

GPIO-pinnien ohjaus on mahdollistettu Raspbian-käyttöjärjestelmässä hallintatiedostoilla, joiden kautta pinnejä voidaan lukea ja ohjata. Pinni otetaan ensin käyttöön kirjoittamalla ”export”-tiedostoon valitun pinnin numero, jonka jälkeen asetetaan pinnin suunta, eli käytetäänkö pinniä sisään- vai ulostulona. Kun suunta on valittu, voidaan pinnin ”value”-tiedostoon joko kirjoittaa ulostulon, tai lukea sisääntulon arvo 0 tai 1. Kun pinniä ei enää käytetä, sen voi vapauttaa kirjoittamalla pinnin numeron ”unexport”-tiedostoon, jonka jälkeen pinnin tilaa ei enää voida hallita tai lukea. [18.]

```
echo 11 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio11/direction
echo 1 > /sys/class/gpio/gpio11/value
echo 0 > /sys/class/gpio/gpio11/value
echo 11 > /sys/class/gpio/unexport
```

Esimerkkikoodi 5. GPIO-pinnin numero 11 käyttäminen ulostulona komentoriviltä.

Esimerkkikoodissa 5 komentorivin kautta otetaan pinni käyttöön kirjoittamalla sen numero ”export”-tiedostoon, valitaan suunta ulostuloksi kirjoittamalla ”direction”-tiedostoon ”out” ja sitten kirjoitetaan ”value”-tiedostoon ulostulon arvoksi ensin 1 ja sitten 0. Lopuksi vapautetaan pinni kirjoittamalla sen numero ”unexport”-tiedostoon. Pinnien koko C++-kielinen ohjausluokka on liitteessä 1.

4.2.3 Kameramoduulin ohjaaminen

Kameramoduulin ohjaaminen oli laiteohjaimen töistä yksinkertaisin toteuttaa. Kameran käyttö ei vaadi erillisiä ajureita tai kirjastoja toimiakseen, vaan Raspbian-käyttöjärjestelmä tarjoaa suorat komentorivikutsut sen ohjaamiseen. Yksittäisen kuvan ottamiseen käytetään ”raspistill”-komentoa, joka ottaa yhden kuvan ja tallentaa sen valitulla nimellä. Kuvan muokkaamisen saa myös tehtyä erilaisilla määrityksillä komentoon, kuten tässä työssä käytetyt vertikaalinen ja horisontaalinen peilaus, koska kameran kaapelin sijainnin takia kamera on asennettu ylösalaisin. Esimerkkikoodissa 6 valinnat ”-hf”, ”horizontal flip”, ”-vf”, ”vertical flip” ja ”-o”, ”output”, jolla tiedostonimi määritellään ja C/C++-koodissa ajettava ”system”-funktio, jolla voidaan kutsua komentorivikomentoja koodin sisällä. Koodissa tiedostonimeksi annetaan Unix-aikaleima, jotta kuvat voidaan erotella toisistaan ja järjestää aikajärjestykseen.

```
system(“raspistill -hf -vf -o imagename.jpg”);
```

Esimerkkikoodi 6. Kameramoduulilla vertikaalisesti ja horisontaalisesti peilatun kuvan ottaminen.

Kameralla voi myös kuvata videoita, joka vaatii ensin videon tallentamisen H.264-enkoodattuna ”raspivid”-komennolla, jonka jälkeen se täytyy muuttaa esimerkiksi selaimen ymmärtämään MP4-muotoon erillisellä ohjelmalla, tässä tapauksessa MP4Boxilla, joka paketoi H.264-raakavideon MP4-muotoon. ”raspivid”-komennossa on lähes samat asetukset kuin ”raspistill”-komennossa, mutta videolle täytyy myös ennalta määritellä videon pituus millisekunteina. Esimerkkikoodissa 7 napataan kymmenen

sekunnin videon ”raspivid”-komennolla, jonka jälkeen tallennettu video paketoidaan MP4Boxilla MP4-muotoon ja alkuperäinen H.264-raakavideo poistetaan.

```
system(“raspivid -t 10000 -hf -vf -o videoname.h264”);
system(“MP4Box -add videoname.h264 videoname.mp4”);
system(“rm videoname.h264”);
```

Esimerkkikoodi 7. Videon nappaaminen kameramoduulilla ja sen paketointi MP4-muotoon.

4.3 Verkko-ohjain

Verkko-ohjain on käyttöliittymän ja laiteohjaimen välissä oleva viestien välittäjä, joka yhdistää sekä laiteohjaimen luomaan Unix-sokettiin, että luo oman web-soketin, johon käyttöliittymä voi sitten yhdistää. Verkko-ohjaimen ainoa tehtävä on välittää laiteohjaimen viestit käyttöliittymälle ja käyttöliittymän viestit takaisin laiteohjaimelle. Verkko-ohjain kirjoitettiin Nodella.

```
var reconnectTimeout = 500;
var reconnects = 0;
var unixSocket = connectToSocket();

…

function connectToSocket() {

    var unixSocket = net.createConnection(socketPath, function () {
        console.log('webserver.js: Connected to Unix socket');
    });

    // Catch all connection errors and recursively try again in intervals
    unixSocket.on('error', function (e) {
        reconnects++;
        // Only try to reconnect 10 times
        if (reconnects > 10) {
            console.log('webserver.js: socket connection timeout, exiting...');
            process.exit();
        } else {
            console.log('webserver.js: error connecting, retrying in ' + recon-
nectTimeout + 'ms...');
            setTimeout(connectToSocket, reconnectTimeout);
        }

    });

    return unixSocket;

}
```

Esimerkkikoodi 8. Laiteohjaimen luomaan Unix-sokettiin yhdistäminen

Esimerkkikoodissa 8 yhdistetään laiteohjaimen luomaan Unix-sokettiin. Koska laiteohjain ja verkko-ohjain käynnistetään skriptillä samanaikaisesti, sokettia ei aina ehditä luoda ennen kuin verkko-ohjain yrittää jo yhdistää siihen. Tästä syystä verkko-ohjaimessa napataan yhdistäessä virheet ja yritetään puolen sekunnin välein uudestaan, kunnes yhteys onnistuu tai yhteyttä on yritetty muodostaa 10 kertaa. [19.]

Kun Unix-sokettiin on saatu yhteys, ohjelmassa luodaan web-soketti, johon käyttöliittymästä voidaan yhdistää.

```
var webSocketServer = ws.createServer(function (conn) {
    console.log('webserver.js: New websocket connection');

    conn.on('text', function (data) {
        clientMessageHandler(data);
    });

    conn.on('close', function (code, reason) {
        console.log('webserver.js: Client disconnected (' + reason + ')');
    });

    process.on('uncaughtException', function (e) {
        console.error(e.stack);
        console.log("webserver.js: Node NOT Exiting...");
    });

}).listen(8001);

// Handle all incoming messages from client (front end)
function clientMessageHandler(msg) {
    unixSocket.write(msg);
}
```

Esimerkkikoodi 9. Web-soketin luominen verkko-ohjaimessa

Esimerkkikoodissa 9 luodaan web-soketti, joka kuuntelee yhteyksiä portista 8001. web-soketissa sidotaan ”text”-tapahtuma funktioon, joka kirjoittaa saapuneen viestin Unix-sokettiin, josta laiteohjain voi sitten lukea sen. ”Text”-tapahtuma laukaistaan, kun web-sokettiin kirjoitetaan käyttöliittymän päässä dataa. Lisäksi koko verkko-ohjaimen prosessin nappaamattomissa virhetilanteissa laukaistavaan ”uncaughtException”-tapahtumaan sidotaan virheviestin tulostaminen. Tapahtuman sitominen estää ohjelman sulkemisen virhetilanteissa. [20; 21.]

```
// Broadcast a message to all connected clients (frontend)
function broadcast(webSocketServer, msg) {
    webSocketServer.connections.forEach(function (conn) {
        conn.sendText(msg);
    });
}

// Event for catching incoming data from sensorapp
unixSocket.on('data', function (data) {
    data = data.toString();
    broadcast(webSocketServer, data);
});
```

Esimerkkikoodi 10. Unix-sokettiin kirjoitettujen viestien välittäminen edelleen käyttöliittymälle

Esimerkkikoodissa 10 sidotaan Unix-soketin ”data”-tapahtuma funktioon, joka lähettää saapuneen viestin kaikille web-sokettiin yhdistettyihin käyttöliittymiin. Unix-soketin ”data”-tapahtuma vastaa web-soketin ”text”-tapahtumaa, joka laukaistaan, kun sokettiin kirjoitetaan dataa. Tämä mahdollistaa usean käyttöliittymän käytön samanaikaisesti.

4.4 Käyttöliittymä

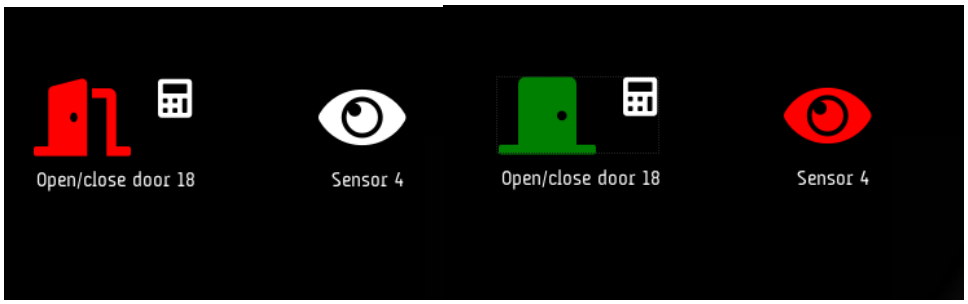
Käyttöliittymä visualisoi laiteohjaimen lähettämän datan käyttäjälle ja lähettää käyttäjän komennot takaisin laiteohjaimelle verkko-ohjaimen kautta.

Käyttöliittymä on selainpohjainen ja koostuu HTML-dokumentista, CSS-tyylitiedostoista ja Javascript-koodista. Käyttöliittymän hallintaan käytetään myös jQuery-kirjastoa, jolla elementtien luominen ja muuttaminen on helpompaa kuin pelkällä Javascriptillä. Käyttöliittymä on ”single-page application”, eli yhden sivun applikaatio, jossa käyttäjää ei ikinä ohjata toiselle sivulle, vaan sivun sisältöä muutetaan ja ohjataan Javascriptillä selaimessa.

Selainpohjainen käyttöliittymä helpottaa päivitysten ajamisen useammalle käyttäjälle samanaikaisesti, kun kaikki ohjelmistologiikka on vain yhdellä palvelimella. Timeconissa taas ohjelmisto on työpöytäapplikaatio, joka voi olla käytössä useammalla koneella samanaikaisesti, jolloin ohjelmistoa päivittäessä täytyy jokaisella käytössä olevalla työpöydällä käydä päivittämässä applikaatio.

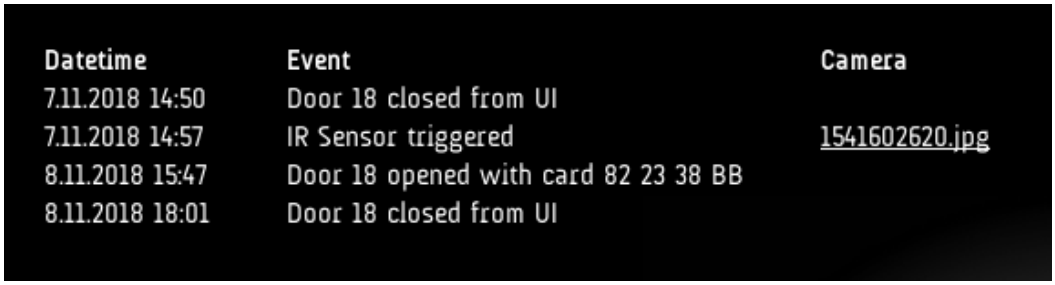
Käyttöliittymän suoritus alkaa web-sokettiin yhdistämisellä, koska ilman yhteyttä ohjelmistoa ei voida käyttää. Niin kauan, kuin yhteyttä ei ole muodostettuna tai yhteys katkeaa kesken käytön, käyttöliittymässä näytetään virheviesti ja painike, jolla yhteyden

muodostusta voidaan yrittää uudelleen. Kun yhteys on muodostettu, käyttöliittymän painikkeiden painallustapahtumiin sidotaan funktioita, jotka lähettävät ennalta määriteltyjä viestejä web-soketille. Liiketunnistinta ei voida käyttöliittymästä ohjata mitenkään, vaan se vain näyttää liiketunnistimen tilan käyttöliittymässä.



Kuva 11. Kuva käyttöliittymän painikkeista. Kuvassa on avoin ja suljettu ovi, sekä hälyttävä ja normaalitilassa oleva liiketunnistin.

Lokitapahtumat pyydetään verkko-ohjaimen kautta laiteohjaimelta tietokannasta ja kirjoitetaan käyttöliittymään lokitapahtumiin. Uusien lokitapahtumien syntyessä laiteohjain lähettää verkko-ohjaimen kautta käyttöliittymälle tapahtuman tiedot JSON-muodossa, jonka käyttöliittymä parsii ja lisää lokitapahtumien listaan käyttöliittymässä. Näin loki pysyy ajan tasalla, eikä tapahtumia tarvitse erikseen kysellä ja hakea tietokannasta useita kertoja, joka säästää tietokannan resursseja. Kaikki verkko-ohjaimen viestit lähetetään kaikille web-sokettiin yhdistetyille käyttöliittymille, jolloin useamman käyttäjän tilanteessa jokainen erillinen käyttöliittymä pysyy ajan tasalla.



Kuva 12. Kuva käyttöliittymän lokista.

Käyttöliittymän Javascript-koodit on jaettu kahteen osaan, joista toinen hoitaa vain käyttöliittymän ulkoasua ja toinen yhteyksiä verkko-ohjaimelle.

5 Kehitystyökalut ja -ympäristö

Työn ohjelmointivaiheessa oli tärkeää, että kehittäminen on sujuvaa ja nopeaa. Tästä syystä ohjelmoitaessa käytetään koodin rakentamisen automaatioon, rakennetun koodin lähettämiseen palvelimelle ja ohjelmiston ajamiseen erilaisia työkaluja kehittäjän työn helpottamiseksi.

5.1 GNU Compiler Collection

GNU Compiler Collection, tai yleisemmin GCC, on C- ja C++-kielien kääntäjä, joka kääntää lähdekoodin konekielisiksi suoritettaviksi ohjelmiksi. GCC:hen kuuluu kääntäjät usealle eri kielelle, kuten C (gcc), C++ (g++), Objective-C, Objective-C++, Fortran (gfortran), Ada (GNAT) ja Go (gccgo). Käytin tässä projektissa GCC:n versiota G++, joka on nimenomaan C++ kielelle tarkoitettu kääntäjä. [22.]

5.2 Node Package Manager

Node Package Manager, eli npm, on nimensä mukaisesti Nodea käyttävä pakettien hallintaohjelmisto. Npm on Noden oletus pakettien hallinta, jolla voidaan hallita ohjelmiston riippuvuuksia ja ladata kehitystyökaluja. Npm sisältää verkossa olevan tietokannan, jossa ylläpidetään sekä julkisia että maksullisia yksityisiä paketteja. Pakettien asentaminen tapahtuu luomalla package.json-tiedosto, jossa määritellään vaadittavat paketit mukaan lukien haluttu versio tai versioskaala paketista.

```
{
    "name": "RasPi",
    "author": "Eetu Purtonen",
    "private": true,
    "version": "0.0.1",
    "devDependencies": {
        "gulp": "*",
        "gulp-autoprefixer": "*",
        "gulp-concat": "^2.6.0",
        "gulp-notify": "*",
        "gulp-plumber": "^1.0.1",
        "gulp-rename": "*",
        "gulp-sass": "^2.0.4",
        "gulp-compass": "*",
        "gulp-sourcemaps": "^1.5.2",
        "nodejs-websocket": "*"
    },
    "repository": {
        "type": "git",
        "url": "https://github.com/purtonen/Raspi-AccessControl"
    }
}
```

Esimerkkikoodi 11. Käyttöliittymäosuuden käyttämä package.json

Package.json-tiedostossa voidaan määritellä ohjelman nimi, kehittäjä, julkisuus, versio, riippuvuudet ja pakettivaraston sijainti. Riippuvuuksien pakettien versionumerot voidaan määritellä usealla eri jokerimerkillä, kuten esimerkiksi

- *: Mikä tahansa versio, lataa aina uusimman saatavilla olevan version
- ^1.x.x: Vähintään versio 1.x.x, mutta kuitenkin pienempi kuin 2.0.0
- ~1.2.x: Vähintään versio 1.2.x, mutta kuitenkin pienempi kuin 1.3.0
- <=1.2.x: Korkeintaan ja mukaan lukien versio 1.2.x
- <1.2.x: Korkeintaan mukaan lukematta versio 1.2.x. [23]

Tämän projektin node-ohjelmassa oli riippuvuutena vain ”nodejs-websocket”, joka mahdollistaa web-sokettien käyttämisen käyttöliittymän yhdistämiseen. Lisäksi käytin käyttöliittymän kehityksessä ”Gulp”-pakettia, joka kääntää Sass-tyylitiedostot selaimen ymmärtämiksi CSS-tiedostoiksi. Gulpilla on myös mahdollista supistaa suurempia Javascript- ja Sass/CSS-tiedostoja, jotta selaimet saavat ladattua ne nopeammin. Tässä projektissa CSS- ja Javascript-tiedostot olivat kuitenkin niin pieniä ja lataaminen tapahtuu saman lähiverkon sisällä, joten supistamista ei tarvittu.

5.3 Make

C++-ohjelman rakentamiseen käytin Make-työkalua, jonka tarkoituksena on automatisoida ohjelmiston rakennusprosessia ajamalla ennalta määriteltyjä komentoja sen sijaan, että kehittäjä ajaisi näitä joka kerta itse manuaalisesti. Maken määrittely tapahtuu sille tarkoitetussa erityisessä tiedostossa nimeltä ”Makefile”.

```
# Compiler
CC = g++
OPTS = -c -Wall

# Project name
PROJECT = sensorapp

# Directories
OBJDIR = sensorapp/obj
SRCDIR = sensorapp/src
BINDIR = sensorapp/bin

# Libraries
LIBS = -pthread -lsqlite3 -lbcm2835

# Files and folders
SRCS    = $(shell find $(SRCDIR) -name '*.cpp')
OBJS    = $(patsubst $(SRCDIR)/%.cpp,$(OBJDIR)/%.o,$(SRCS))

# Targets
$(PROJECT): buildrepo $(OBJS) npm
    $(CC) $(OBJS) $(LIBS) -o $(BINDIR)/$@

$(OBJDIR)/%.o: $(SRCDIR)/%.cpp
    $(CC) $(OPTS) -c $< -o $@

npm:
    npm install

clean:
    rm $(OBJDIR) -rf

buildrepo:
    @$(call make-repo)

# Create obj and bin directory structure
define make-repo
    mkdir -p $(OBJDIR)
    mkdir -p $(BINDIR)
    for dir in $(SRCDIRS); \
    do \
        mkdir -p $(OBJDIR)/$$dir; \
    done
endef
```

Esimerkkikoodi 12. Projektissa käyttämäni Makefilen sisältö.

Make ajetaan komentoriviltä komennolla ”make”, joka suorittaa Makefilessä määritellyn oletustoiminnon. Tässä tapauksessa etsitään kaikki tiedostopäätteellä ”.cpp” olevat

lähdetiedostot SRCDIR-muuttujassa määritellystä kansiosta, ajetaan g++:lla lähdetiedostojen käännös objektitiedostoiksi OBJDIR-muuttujassa määriteltyyn kansioon ja lopuksi linkitetään g++:lla luodut objektitiedostot ja LIBS-muuttujassa määritellyt ulkoiset kirjastot yhteen suoritettavaan binääritiedostoon, joka viedään BINDIR-muuttujassa määriteltyyn kansioon.

Tämän lisäksi ajetaan vielä ”npm install”, joka lataa package.json-tiedostossa määritellyt node-ohjelman riippuvuudet. Noden riippuvuuksien päivittäminen ei olisi aina välttämätöntä, varsinkaan tässä projektissa, koska node-ohjelmalla on vain yksi riippuvuus, eikä se muuttunut kertaakaan koko kehityksen aikana. Päätin kuitenkin sisällyttää NPM:n asennuskomennon rakennusvaiheeseen, jotta ohjelmiston voisi myös helposti siirtää toiseen ympäristöön, jossa vaadittuja riippuvuuksia ei ole ja asentaa sitten vain yhdellä komennolla.

5.4 Käynnistys- ja lopetusskriptit

Koska järjestelmässä on useampi toisistaan riippuvainen suoritettava ohjelma, päätin työnkulun helpottamiseksi kirjoittaa sekä käynnistämiseen että sulkemiseen erittäin minimaaliset bash-skriptit.

```
#!/bin/bash

./sensorapp/bin/sensorapp &
node webserver.js &
```

Esimerkkikoodi 13. Käynnistysskripti

Käynnistysskripti käynnistää c++-ohjelman binääritiedoston, vapauttaa komentorivin, käynnistää node-ohjelman ja vapauttaa taas komentorivin.

```
#!/bin/bash

killall sensorapp node
```

Esimerkkikoodi 14. Lopetusskripti

Lopetusskripti vain tappaa käynnistysskriptin käynnistämät prosessit.

Molemmissa skripteissä ensimmäisellä rivillä määritellään erityisellä ”shebang”-nimisellä kommentilla skriptin suorittamiseen käytettävä komentotulkki, joka tässä tapauksessa on Bash.

5.5 Kehitysympäristö

Projektin kehitys tapahtui omalla tietokoneella, mutta GPIO-pinnien ja Raspberry Pin kameramoduulin käytön takia ohjelmiston suorittaminen ja testaaminen täytyi tehdä Raspberry Pillä. Projektin koodit täytyi siis lähettää jollain tavalla Raspberry Pille. Tähän tarkoitukseen olisi monia eri työkaluja, kuten esimerkiksi Githubin ja Gitlabin automaattinen koodin lähettäminen jokaisen uuden Git commitin saapuessa. Näiden työkalujen käyttäminen kuitenkin vaatisi kohdejärjestelmässä jonkin verran konfigurointia.

Koska kehittäminen tapahtui omalla kotikoneella ja kohdejärjestelmä, eli Raspberry Pi, sijaitsi samassa verkossa, eikä internetin yli koodin lähettäminen ollut tarpeellista, päädyin yksinkertaisempaan ratkaisuun, eli tiedostojen siirtoon SFTP-yhteydellä. Jokaisen muutoksen jälkeen tiedostojen siirtäminen manuaalisesti ei kuitenkaan ole hyvä eikä käytännöllinen ratkaisu, vaan prosessi vaatii jonkinlaista automaatiota. Koska kotikoneellani ja Raspberry Pillä on molemmilla käytössä Linux, helpoin ratkaisu oli kehityskoneella kansion kiinnittäminen verkon yli kohdekoneella olevaan kansioon SSHFS:n avulla. Kun kansio on kiinnitetty, se käyttäytyy kuin käyttäjä olisi suoraan kohdejärjestelmän sisällä ja kaikki muutokset tiedostoihin tapahtuu reaaliajassa sekä kehitys- että kohdekoneella. [25.]

SSHFS on vain tiedostojärjestelmä, joten komentorivikomentojen ajaminen kehityskoneella kiinnitetyssä kansiossa suorittaa komennot edelleen kehityskoneella. Tämä tarkoittaa myös sitä, että projektin rakentaminen Makella voidaan tehdä kehityskoneella, jolloin ainakin C++-ohjelman kääntäminen näytti tapahtuvan hieman nopeammin koneiden tehoeroista johtuen. Maken ajaminen kehityskoneella kuitenkin vaatii, että kehityskoneella on Make ja npm asennettuna.

Varsinaisen rakennetun ohjelmiston ajaminen täytyy kuitenkin edelleen tehdä Raspberry Pillä, jotta sen GPIO-pinnejä voidaan ohjata ja lukea ohjelmalla. Tätä varten tarvitaan vielä erillinen SSH-yhteys kohdekoneelle, jossa käynnistys- ja lopetusskriptit voidaan

sitten ajaa. Yhteyden yksinkertaistamiseksi loin kehityskoneelle komentorivialiaksen, joka ajaa ssh-komennon, joka avaa yhteyden kohdekoneelle. Lisäksi lisäsin kehityskoneen SSH-avaimen kohdekoneen sallittujen avaimien listaan, jotta kehityskoneelta voi avata SSH-yhteyden kohdekoneelle ilman salasanaa.

6 Yhteenveto ja jatkokehitys

Työn tarkoituksena oli rakentaa avoin kulunvalvontajärjestelmän prototyyppi, jolla voi ohjata lukkoa lukijan ja käyttöliittymän kautta, valvoa tilaa liiketunnistimen avulla ja tallentaa kuvia kameralla. Prototyypiksi työ oli onnistunut, mutta jatkokehitykselle on vielä paljon tilaa.

Suurimmat haasteet projektissa oli laiteohjaimen ohjelmoinnissa, erityisesti lukijan ohjaaminen osoittautui vaikeaksi ajureiden yhteensopivuus- ja versio-ongelmien takia. Myös GPIO-pinnien numeroinnin epäjohdonmukaisuus aiheutti hämmennystä kytkentä- ja ohjelmointivaiheessa. Pinneillä on käytännössä kaksi eri numeroa, joista toinen on fyysisen pinnin sijainti Raspberry Pi:n piirilevyllä ja toinen on pinnin tunnistenumero, jolla pinniä ohjataan tai luetaan. Esimerkiksi pinni numero 12 on tunnistenumerolla GPIO18.

Uusina asioina minulle tuli projektissa prosessien välinen kommunikointi Unix-sokettien avulla ja laiteohjelmointi. Kielissä C++ oli myös suhteellisen tuntematon minulle, mutta se tuli nopeasti työn aikana tutuksi. Lähinnä oppimistarkoitukseen tehtynä työ oli erittäin onnistunut ja koin oppineeni paljon uutta laiteohjelmoinnista, johon en koulussa tai töissä ole niin paljon päässyt tutustumaan.

Koska projekti oli lähinnä prototyypiksi tarkoitettu, jatkokehitykseen jää vielä paljon tekemistä. Uusia ominaisuuksia ohjelmistoon tulisi ainakin:

- Järjestelmässä tulisi olla useampia sensoreita ja ovia ja niiden lisäämisen tulisi olla mahdollista käyttöliittymän kautta.
- Korttien rekisteröinti tulisi olla mahdollista käyttöliittymän kautta, esimerkiksi luettamalla kortti lukijaan, jolloin käyttöliittymän lokiin tulisi merkintä uudesta kortista, ja sitä kautta kortin saisi lisättyä tietokantaan ja määriteltyä kortille oikeuksia eri oviin.
- Järjestelmässä tulisi olla tuki useammalle eri sensorityyppille, eli esimerkiksi ovien tilaa voisi valvoa magneettikoskettimilla ja lasirikkoilmaisimia, jotka antavat hälytyksen havaitessaan lasin rikkoutumisen äänen tai paine-eron perusteella.
- Sensorien lisäämisen myötä GPIO-pinnejä tarvitsisi myös lisää, esimerkiksi laajennuskorteilla, jotka täytyisi ohjelmoida ja konfiguroida.
- Käyttöliittymän ominaisuuksien kasvaessa käyttöliittymään voisi olla hyödyllistä käyttää jotakin ohjelmistokehystä, kuten esimerkiksi Angular, React tai Vue.js.

Uusien ominaisuuksien lisäksi koodikannan kasvaessa koodin organisointia ja luettavuutta voisi vielä parantaa.

Lähteet

1 What is a Raspberry Pi? 2017. Verkkodokumentti. Raspberry Pi Foundation. https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/. Luettu 1.12.2017.

2 RPi hardware history. 2018. Verkkodokumentti. eLinux. https://elinux.org/RPi_HardwareHistory. Luettu 1.9.2018.

3 GPIO. 2018. Verkkodokumentti. Raspberry Pi Foundation. https://www.raspberrypi.org/documentation/usage/gpio/. Luettu 3.6.2018.

4 Camera module. 2018. Verkkodokumentti. Raspberry Pi Foundation. https://www.raspberrypi.org/documentation/usage/camera/raspicam/README.md. Luettu 8.5.2018.

5 What is RFID and how does RFID work? 2018. Verkkodokumentti. AB&R. https://www.abr.com/what-is-rfid-how-does-rfid-work/. Luettu 4.10.2018.

6 Control high voltage devices – Arduino Relay tutorial. 2018. Verkkodokumentti. How to mechatronics - Dejan. https://howtomechatronics.com/tutorials/arduino/control-high-voltage-devices-arduino-relay-tutorial/. Luettu 14.7.2018.

7 RC522 on the Raspberry Pi. 17.2.2017. Verkkodokumentti. Hawry. https://dev.hawry.net/rc522-on-the-raspberry-pi-without-python/. Luettu 2.6.2018.

8 Python. 2017. Verkkodokumentti. Raspberry Pi Foundation. https://www.raspberrypi.org/documentation/usage/python/. Luettu 1.12.2017.

9 What exactly is Node.js? 18.4.2018. Verkkodokumentti. Priyesh Patel. https://medium.freecodecamp.org/what-exactly-is-node-js-ae36e97449f5. Luettu 20.9.2018.

10 When to use what language and why. 14.10.2010. Verkkodokumentti. Seraphimsan. http://www.cplusplus.com/articles/42E1wA7f/. Luettu 4.12.2017.

11 PHP or Python: best language for werver side development. 11.4.2017. Verkkodokumentti. Max Hornostaiev. https://www.upwork.com/hiring/development/php-or-python-for-server-side-development/. Luettu 15.1.2018.

12 Django documentation. 2018. Verkkodokumentti. Django Software Foundation. https://docs.djangoproject.com/en/2.1/. Luettu 12.2.2018.

13 Tornado documentation. 2018. Verkkodokumentti. Tornado Web. https://www.tornadoweb.org/en/stable/. 14.3.2018.

14 Unix domain sockets vs. internet sockets. 25.2.2005. Verkkodokumentti. Robert Watson. https://lists.freebsd.org/pipermail/freebsd-performance/2005-February/001143.html. Luettu 2.6.2018.

15 Socket. 2018. Verkkodokumentti. The Open Group. https://linux.die.net/man/3/socket. Luettu 1.7.2018.

16 Sockets tutorial. 2018. Verkkodokumentti. Robert Ingalls. http://www.cs.rpi.edu/~moorthy/Courses/os98/Pgms/socket.html. Luettu 12.8.2018.

17 RPi-RFID. 13.6.2018. Verkkodokumentti. Paguz. https://github.com/paguz/RPi-RFID. Luettu 15.9.2018.

18 Introduction to accessing the Raspberry Pi GPIO pins in C++ (sysfs). Julkaistu 18.11.2012, päivitetty 26.8.2013. Verkkodokumentti. Hertaville. http://hertaville.com/introduction-to-accessing-the-raspberry-pis-gpio-in-c.html. Luettu 1.12.2017.

19 Node.js documentation – net.createConnection. 2018. Verkkodokumentti. Joyent. https://nodejs.org/api/net.html#net_net_createconnection. Luettu 1.3.2018.

20 Nodejs-websocket documentation. 2016. Verkkodokumentti. Sitegui. https://www.npmjs.com/package/nodejs-websocket. Luettu 2.3.2018.

21 Node.js error handling. 2017. Verkkodokumentti. Joyent. https://www.joyent.com/node-js/production/design/errors. Luettu 27.4.2018.

22 GCC, the GNU Compiler Collection. Muokattu 26.10.2018. Verkkodokumentti. GNU Project. https://gcc.gnu.org/. Luettu 15.12.2017.

23 How to use semantic versioning. 2018. Verkkodokumentti. Npm, Inc. https://docs.npmjs.com/getting-started/semantic-versioning. Luettu 25.6.2018.

24 Using make and writing Makefiles. 2018. Verkkodokumentti. Tia Newhall. https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html. Luettu 14.3.2018.

25 SSHFS. 2018. Verkkodokumentti. Libfuse. https://github.com/libfuse/sshfs. Luettu 8.10.2018.

GPIO-pinnien ohjausluokka

```
#include <fstream>
#include <string>
#include <iostream>
#include <sstream>
#include "gpio.h"

using namespace std;

GPIO::GPIO() {
    this->gpionum = "4"; //GPIO4 is default
    this->valueChanged = false;
}

GPIO::GPIO(string gnum) {
    this->gpionum = gnum; //Instantiate GPIO object for GPIO pin number "gnum"
    this->valueChanged = false;
}

int GPIO::export_gpio() {
    string export_str = "/sys/class/gpio/export";
    ofstream exportgpio(export_str.c_str()); // Open "export" file. Convert C++ string
to C string. Required for all Linux pathnames
    if (!exportgpio) {
        cout << " OPERATION FAILED: Unable to export GPIO" << this->gpionum << " ." <<
endl;
        return -1;
    }

    exportgpio << this->gpionum; //write GPIO number to export
    exportgpio.close(); //close export file
    return 0;
}

int GPIO::unexport_gpio() {
    string unexport_str = "/sys/class/gpio/unexport";
    ofstream unexportgpio(unexport_str.c_str()); //Open unexport file
    if (!unexportgpio) {
        cout << " OPERATION FAILED: Unable to unexport GPIO" << this->gpionum << " ." <<
endl;
        return -1;
    }

    unexportgpio << this->gpionum; //write GPIO number to unexport
    unexportgpio.close(); //close unexport file
    return 0;
}

int GPIO::setdir_gpio(string dir) {

    string setdir_str = "/sys/class/gpio/gpio" + this->gpionum + "/direction";
    ofstream setdirgpio(setdir_str.c_str()); // open direction file for gpio
    if (!setdirgpio) {
        cout << " OPERATION FAILED: Unable to set direction of GPIO" << this->gpionum <<
" ." << endl;
        return -1;
    }

    setdirgpio << dir; //write direction to direction file
```

```
    setdirgpio.close(); // close direction file
    this->direction = dir;
    return 0;
}

int GPIO::setval_gpio(string val) {

    string setval_str = "/sys/class/gpio/gpio" + this->gpionum + "/value";
    ofstream setvalgpio(setval_str.c_str()); // open value file for gpio
    if (!setvalgpio) {
        cout << " OPERATION FAILED: Unable to set the value of GPIO" << this->gpionum <<
" ." << endl;
        return -1;
    }

    setvalgpio << val; //write value to value file
    setvalgpio.close(); // close value file

    if (this->previousValue != val) {
        this->valueChanged = true;
        this->previousValue = val;
    }

    return 0;
}

int GPIO::getval_gpio(string &val) {
    string line;

    string getval_str = "/sys/class/gpio/gpio" + this->gpionum + "/value";
    ifstream getvalgpio(getval_str.c_str(), ifstream::in); // open value file for gpio
    if (!getvalgpio) {
        cout << " OPERATION FAILED: Unable to get value of GPIO" << this->gpionum << "
." << endl;
        return -1;
    }

    getvalgpio >> val; //read gpio value

    if (val != "0")
        val = "1";
    else
        val = "0";

    getvalgpio.close(); //close the value file

    if (this->previousValue != val) {
        this->valueChanged = true;
        this->previousValue = val;
    }

    return 0;
}

string GPIO::get_gpionum() {

    return this->gpionum;
}

bool GPIO::valueIsChanged() {
    return this->valueChanged;
}

void GPIO::resetValueChanged() {
```

```
    this->valueChanged = false;
}
```