Aki Foudila

Eläimen punnituspiste eläinklinikalla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Automaatiotekniikka

Insinöörityö

12.5.2016

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Aki Foudila Eläimen punnituspiste eläinklinikalla 34 sivua 12.5.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Automaatiotekniikka
Suuntautumisvaihtoehto	
Ohjaaja(t)	Toimitusjohtaja Janne Huttunen Lehtori Antti Liljaniemi

Opinnäytetyössä toteutettiin Finnish Net Solutions -nimiselle ohjelmistoyritykselle järjestelmä, jolla eläinklinikoiden asiakkaat voivat lemmikkinsä mikrosirun perusteella tallentaa lemmikin painon eläinklinikan Provet-toiminnanohjausjärjestelmään. Järjestelmällä helpotetaan eläinklinikan vastaanottotyöntekijöiden työtaakkaa, sekä tuotetaan Finnish Net Solutionsille uusi myytävä lisäpalvelu Provet-ohjelmiston käyttäjille.

Projektissa liitettiin Raspberry Pi -mikrotietokoneeseen mikrosirunlukija, puntari, sekä kosketusnäyttö. Sovellus Raspberry Pi -tietokoneelle ohjelmoitiin Python-ohjelmointikielellä ja kosketusnäytön käyttöliittymän luonnissa käytettiin GTK+-käyttöliittymäkirjastoa. Yhteydet Provet-järjestelmiin toteutettiin rajapinnan kautta, jota varten työhön tehtiin testirajapinta PHP-ohjelmointikielellä ja Lumen-mikroverkkokehitysrungolla.

Opinnäytetyön lopputuloksena syntyi toimiva kokonaisuus eläimen punnitsemiseksi ja painon lähettämiseksi Provet-ohjelmistoon rajapinnan kautta. Projektin aikataulun puitteissa toteutukseen ei ehditty tekemään sovelluksen etäpäivitystoimintoa, joten projektia ei saatu täysin tuotantovalmiiksi.

Punnituspisteellä saadaan eläinklinikoiden vastaanottotyöntekijöiden työtaakkaa vähennettyä, sekä tuotettua Finnish Net Solutionsille myytävä lisäpalvelu Provet-ohjelmistojen käyttäjille. Projektin kehitystä jatketaan tuontantovalmiiksi ja sille löytyy jo testikäyttäjiä. Kiinnostusta painonpunnituspisteen käyttöönottoon on osoitettu jo muun muassa eräältä Ruotsalaiselta eläinklinikalta.

Avainsanat	Raspberry Pi, rajapinta, sarjaporttilaitteet, ohjelmointi, GTK+

Author(s) Title Number of Pages Date	Aki Foudila Pet Weighing System at Animal Clinic 34 pages 12 May 2016
Degree	Bachelor of Engineering
Degree Programme	Automation Technology
Specialisation option	
Instructor(s)	Janne Huttunen, CEO Antti Liljaniemi, Senior Lecturer

This thesis was commissioned by a software company called Finnish Net Solutions. The goal of this thesis was to create a system which the clients of animal clinics can use to record their pet's weight in the clinic's Provet practice management system. To record the weight for the correct patient in the database of the Provet system, the patient's microchip implant is used for identification. This reduces the workload of the clinic reception workers and produces Finnish Net Solutions a new product it can sell to the users of their Provet systems.

A microchip scanner, scale and a touch screen were connected to a Raspberry Pi microcomputer. The software in Raspberry Pi was programmed in Python programming language and the graphical user interface was developed using GTK+ library. API was created to connect from the Raspberry Pi to the Provet practice management systems. To test and define the API, a test server was developed using PHP programming language and Lumen micro-framework.

As a result of this project, a system was produced that is capable of identifying the pet and recording its weight in a Provet system through an API. To test and define the API, a test server was successfully created. Due to time constraints, the project did not reach a production-ready state, as it was missing the essential functionality to remotely update the software in the Raspberry Pi.

The product created during this study reduces the workload of animal clinic reception workers and produces a new product Finnish Net Solutions can provide to the users of Provet. The project will be developed further to reach a production-ready state and test users have already been found. A Swedish animal clinic has also shown interest in the system.

Keywords	Raspberry Pi, API, serial port devices, programming, GTK+

Sisällys

Lyhenteet

CSS	Cascading Style Sheets. WWW-dokumenteille kehitetty tyylien säännöstö.
FNS	Finnish Net Solutions. Yritys jolle opinnäytetyö tehtiin.
GTK+	GIMP Toolkit. C-ohjelmointikielellä luotu avoimen lähdekoodin kirjasto graafisten käyttöliittymien luontiin.
HTML	Hypertext Markup Language. Verkkodokumenttien rakenteen kuvauskieli.
HTTP	Hypertext Transfer Protocol. Protokolla tiedon siirtoon asiakkaan ja palvelimen välillä. Käytetään WWW-dokumenttidatan siirtoon.
MAC	Media Access Control. Verkkolaitteeseen valmistusvaiheessa määritetty yksilöllinen osoite.
NOOBS	New Out Of Box Software. Raspberry Pi -tietokoneelle suunniteltu käyttöjärjestelmien asennusohjelma.
RS-232	Recommended Standard 232. Tietokonenlaitteiden väliseen kommunikointiin tarkoitettu tietoliikenneportti. Tietokoneissa paremmin tunnettu nimellä sarjaportti.
SSH	Secure Shell. Protokolla salatun yhteyden muodostamiseen laitteiden välillä.

1 Johdanto

Tämä insinöörityö tehdään Finnish Net Solutions (FNS) -nimiselle ohjelmistoyritykselle, joka toteuttaa räätälöityjä verkkopalveluja, sekä kehittää ja myy eläinklinikoiden ja terapia-alojen hallintajärjestelmiä. Työ liittyy Provet-tuoteperheen selainpohjaisiin eläinklinikoiden toiminnanohjausjärjestelmiin Provet Cloud ja Provet Net.

Eläinklinikoilla eläimet punnitaan asiakkaan saapuessa, jotta tarvittaessa oikea määrä lääkettä osataan määrätä. Eläin punnitaan aulassa ja vastaanottotyöntekijä kirjaa painon Provet-ohjelmistoon. Työn tarkoituksena on toteuttaa eläimen painon tallennus klinikalla skannatun mikrosirun perusteella suoraan eläimen tietoihin, jolloin vastaanottotyöntekijän ei manuaalista tallennusta tarvitse tehdä. Punnituspiste tehostaa klinikoiden vastaanottotoimintaa ja tuo Finnish Net Solutionsille uuden myytävän lisäpalvelun Provet-järjestelmien käyttäjille.

Toteutus ohjelmoidaan Raspberry Pi -pienoistietokoneelle, johon liitetään mikrosirunlukija, puntari, sekä kosketusnäyttö. Näyttöä käytetään asiakkaan ohjeistamiseen prosessissa, sekä painon punnitsemisessa tarvittavien painikkeiden käyttöön. Kokonaisuudelle rakennetaan kotelo, jotta siitä saadaan siistimmän näköinen. Sovellus ohjelmoidaan Python-ohjelmointikielellä, käyttäen GTK+-käyttöliittymäkirjastoa. Laitteet sijoitetaan eläinklinikan aulaan näkyvälle paikalle, jossa asiakas voi itse suorittaa lemmikkinsä painon tallennuksen. Ensin monitorilla ohjeistetaan lemmikin mikrosirun lukemisesta. Kun asiakas on lukenut mikrosirun puntarin vierestä löytyvällä mikrosirunlukijalla, ohjeistetaan asiakas punnitsemaan lemmikki. Puntarilta paino tallennetaan rajapinnan kautta suoraan Provet-ohjelmistoon ja asiakkaalle näytetään punnituksen tulos, sekä lemmikin edellinen paino ja painon muutos, jos eläin on klinikalla aiemmin punnittu.

Työn alussa käydään läpi punnitsemisen prosessi, työn tavoitteet, sekä esitellään käytetyt laitteet, niiden valintaperusteet ja toimintaperiaatteet. Tämän jälkeen käydään läpi sovelluksen teknistä toteutusta, esitellään projektin kehitysmenetelmiä ja kuvataan rajapinta Provet-ohjelmistoihin. Työssä pohditaan myös järjestelmän tietoturvariskejä ja niiden minimointia.

2 Eläimen punnitseminen

2.1 Tavoitteet

Työn käyttäjän kannalta tärkeimmäksi tavoitteeksi asetettiin prosessin pitäminen mahdollisimman yksinkertaisena ja vaatimaan mahdollisimman vähän työtä. Tällä tavoin punnitsemisesta pyritään tekemään helppoa ja nopeaa. Eläimen punnitseminen on asiakkaalle yksinkertainen toimenpide. Liian monimutkaisen näköisessä ja tuntuisessa prosessissa tai käyttöliittymässä on vaara, että asiakas punnitsee eläimen ja käy ilmoittamassa painon tiskillä hyödyntämättä automaattista tallennusta mikrosirun perusteella. Jotta asiakkaan tulee toimintoa käytettyä, tulee sen olla mahdollisimman yksinkertainen ja helposti lähestyttävä. Yksinkertaisella käyttöliittymällä eläinklinikan asiakasta ei tarvitse erikseen opettaa punnituspisteen käytössä. (1.)

Yksinkertaisuuteen pääsemiseksi pyrittiin prosessin vaiheet ja käyttöliittymän painikkeet pitää minimissään. Käyttöliittymässä tavoiteltiin vähäistä tekstimäärää ja selkeää ulkoasua. Täysin painikkeetonta toteutusta harkittiin, mutta tämä todettiin liian vaikeaksi tavoitteeksi painon tallennuksen kohdalla. Teoriassa painon voisi tallentaa esimerkiksi kun se on ollut 5 sekuntia vakaana, mutta virhetallennusten riski arvioitiin liian suureksi. Virhetallennukset ovat asiakkaille turhauttavia ja työntekijöille vaivalloisia korjata, joten käyttöliittymään päätettiin lisätä painike painon tallentamiseksi. Automaattinen tallennus voi lisäksi saada käyttäjän luulemaan tallennuksen olleen käyttäjän tekemä virhe, jos tämä yllättyy painon automaattisesta kirjauksesta (2).

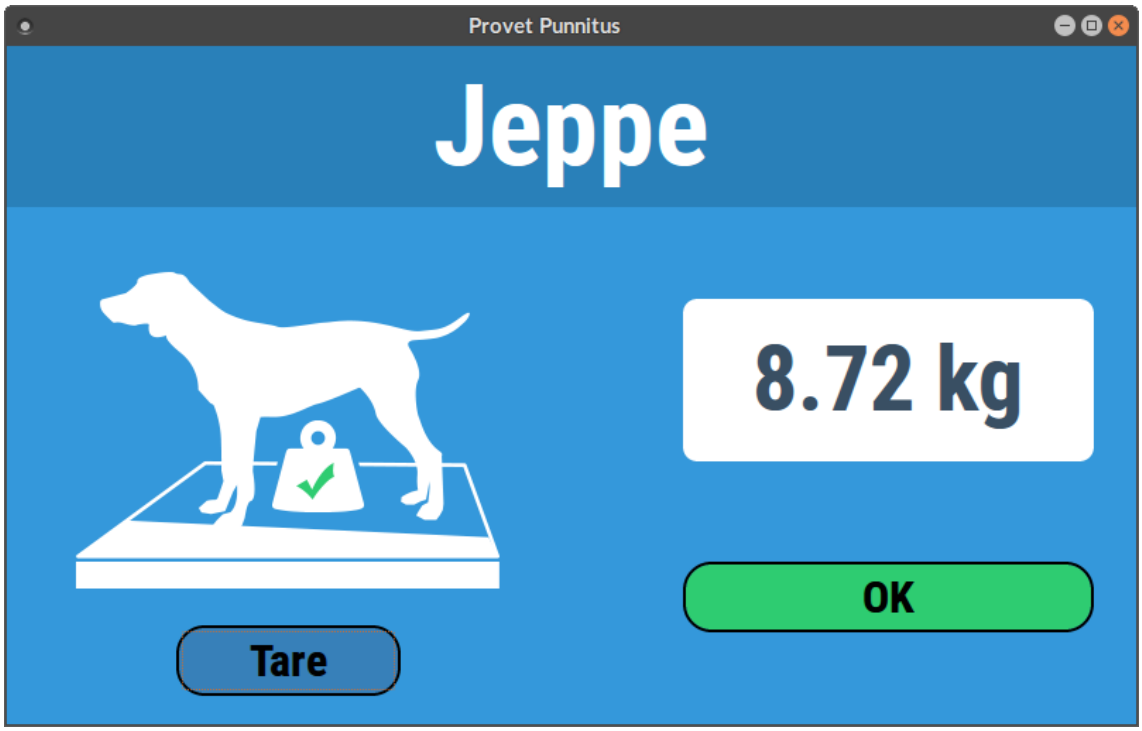
2.2 Prosessi

Puntari, mikrosirunlukija ja kosketusnäyttö sijaitsevat eläinklinikan aulassa näkyvällä paikalla asiakkaiden käytettäväksi. Puntarin ollessa vapaa on näytössä ohjeteksti mikrosirun lukemiseen lukijalla, joka löytyy puntarin vierestä. Näytöllä on tekstin lisäksi kuva, joka ohjaa asiakasta mikrosirun lukemisessa ja punnitsemisessa. Kuvassa 1 nähdään ohjelman ensimmäisen vaiheen ulkoasu.

Kuva 1. Prosessin ensimmäinen vaihe, jossa odotetaan mikrosirun lukua

Asiakkaan käyttäessä lukijaa ja löydettyä mikrosirun, hakee järjestelmä rajapinnan kautta Provet-järjestelmästä mikrosirun numeron perusteella eläimen nimen, sekä edellisen punnituksen päivämäärän ja painon. Jos Provet ilmoittaa, että mikrosirun numerolla ei löytynyt eläintä, ohjeistetaan asiakasta ottamaan yhteys henkilökuntaan. Rekisteröimättömien mikrosirujen tapauksessa henkilökunta joutuu yhdistämään mikrosirun Provet-potilastietokannan eläimeen. Mikäli mikrosiru on rekisteröity Provet-järjestelmässä, siirrytään eläimen punnitsemiseen.

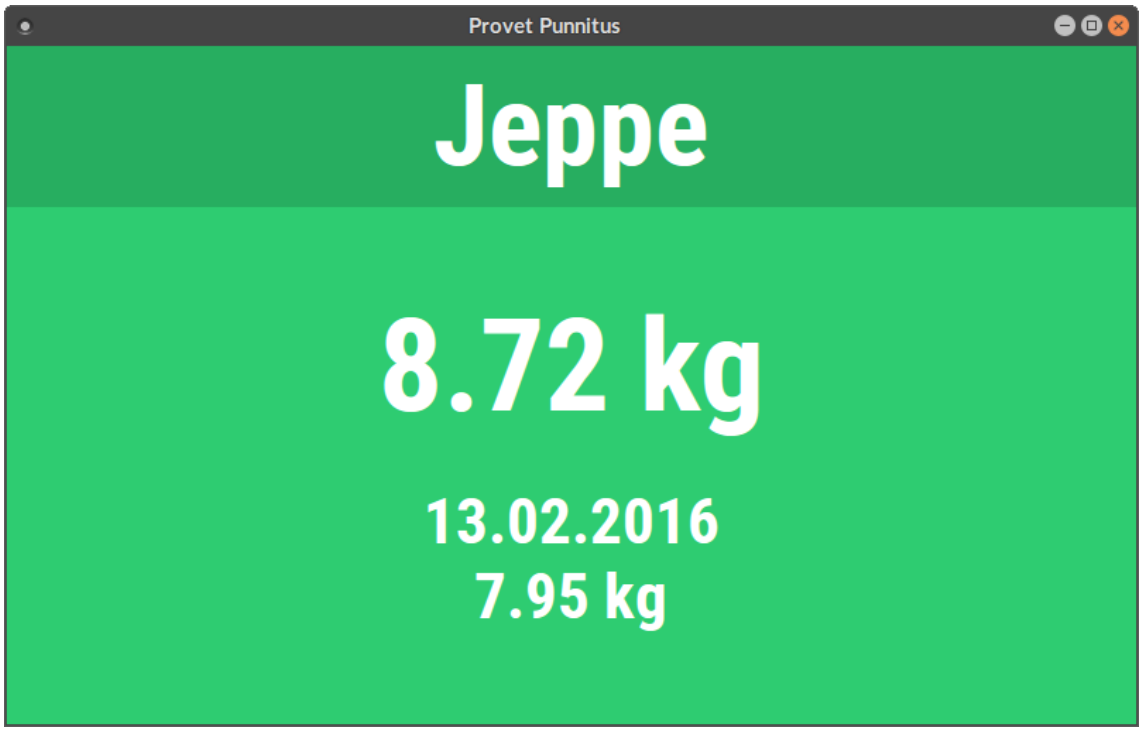
Punnitusvaiheessa ruudulla näytetään tunnistetun eläimen nimi, puntarilta luettu paino, tare-painike puntarin lukeman nollaamiseksi, sekä ok-painike, jolla paino hyväksytään (kuva 2). Tare-painikkeella asiakas voi tarvittaessa nollata painolukeman. Tätä voidaan käyttää, jos lemmikki pitää punnita esimerkiksi laatikossa. Laatikko asetetaan puntarille, jonka jälkeen painetaan tare, joka nollaa lukeman ja lemmikki voidaan laittaa laatikkoon. Puntari näyttää tällöin lemmikin painon. Puntarin lukema hyväksytään painamalla ok-painiketta. Hyväksymisen jälkeen Raspberry Pi lähettää rajapinnan kautta Provet-ohjelmistoon eläimen mikrosirun numeron, sekä puntarilta luetun painon. Näin Provet pystyy tallentamaan uuden painon kyseisen mikrosirun omaavalle eläimelle.



Kuva 2. Eläimen punnitusvaihe

Asiakas saattaa jättää punnitsemisen kesken, jolloin ohjelma täytyy pystyä palauttamaan takaisin ensimmäiseen vaiheeseen. Tähän harkittiin painikkeen lisäystä, mutta käyttöliittymä haluttiin pitää mahdollisimman yksinkertaisena ja painikkeiden määrä mahdollisimman vähäisenä. Ohjelmaan päätettiin lisätä 30 sekunnin ajastin, joka alkaa aina alusta kun puntarin lähettämä painotieto muuttuu. Mikäli paino on muuttumattomana 30 sekunnin ajan, palaa ohjelma ensimmäiseen vaiheeseen odottamaan uuden mikrosirun lukua.

Painon hyväksymisen jälkeen asiakkaalle näytetään yhteenveto punnituksen onnistumisesta. Näkymässä näytetään eläimen nimi ja tallennettu paino. Tässä vaiheessa näytetään myös eläimen aiempi paino ja painon muutos, jos nämä tiedot löytyivät Provetista. Kuvasta 3 nähdään miltä näkymä näyttää ohjelmassa. Tulokset näkyvät minuutin ajan, jonka jälkeen palataan prosessin ensimmäiseen vaiheeseen odottamaan seuraavan mikrosirun lukua.



Kuva 3. Punnituksen tulos ja edellinen paino

3 Käytetyt laitteet

3.1 Mikrosirunlukija

Työssä käytettävä mikrosirunlukija on Scanner Angel -valmistajan lukija nimeltä Halo (kuva 4). Lukija valittiin työhön sen kohtuullisen hinnan, USB-yhteyden ja helppokäyttöisyyden takia. Paristoja ei tarvita, vaan lukija lataa saa virtaa ja lataa akkunsa USB-yhteyden kautta. Lukijassa on vain yksi painike takana, josta lukija herätetään sen mennessä virransäästötilaan 50 sekunnin käyttämättömyyden jälkeen. Lukija tunnistaa mikrosirun automaattisesti sen ollessa tarpeeksi lähellä sirua, jolloin sirun numerosarja näkyy lukijan näytössä. Numerosarjan voi tyhjentää uutta lukua varten painamalla lukijan takana olevaa painiketta. Lukija osaa lukea standardin ISO-11784 ja ISO-11785 mukaisia 15-numeroisia mikrosiruja. Standardi on yleisin Euroopassa käytetty mikrosirutyyppi, mutta on käytössä maailmanlaajuisesti. (3.)

Lukijaa on FNS:llä yritetty lukea tietokoneen kautta aiemminkin, joten ohjeet yhteyden muodostamiseksi löytyivät valmiiksi. Laitteeseen ei ollut virallisia ohjeita, vaan laitevalmistajan mikrosirunlukijoita kehittänyt insinööri oli kirjoittanut ohjeet FNS:lle sähköpostitse.

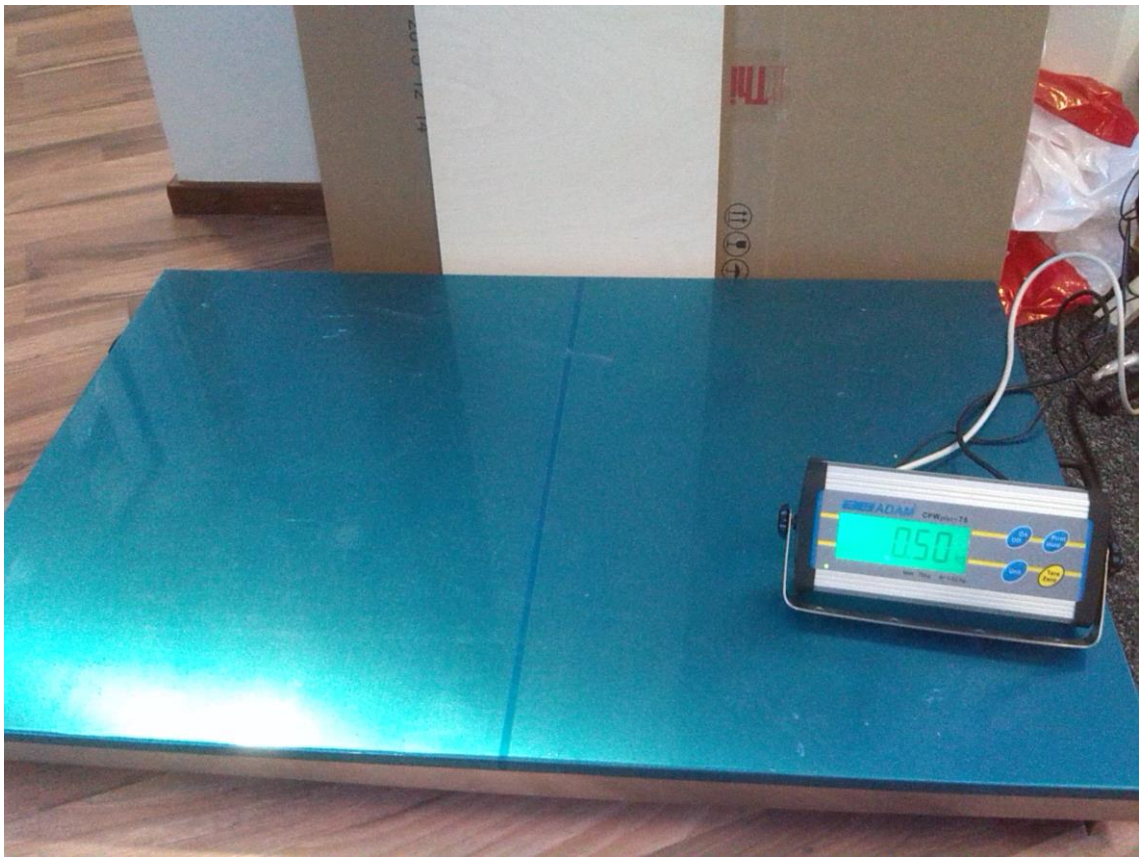
Kuva 4. Halo-mikrosirunlukija (3)

Lukija tukee myös Scanner Angelin kadonneiden lemmikkien tietokantaa, mutta se on käytössä vain Britanniassa, joten toimintoa ei tässä insinöörityössä hyödynnetä. Kadonneiden eläinten mikrosirujen tietokannan päivittäminen lukijaan täytyy tehdä Scanner Angelin oman sovelluksen kautta, joka tukee vain Windows-käyttöjärjestelmää. Koska Raspberryllä käytetään Linux-pohjaista käyttöjärjestelmää, ei tietokannan päivitystä työhön voitu lisätä.

Eläimien sirutuksia on aloitettu pakottamaan jo jonkin verran. Ruotsissa ja Englannissa koirat on pakko siruttaan (4; 5). Eläimen on myös oltava sirutettu, jos sen haluaa viedä EU-maahan (6). Mikäli sirutus vielä yleistyy, tulee työllä jatkossa olemaan enemmän käyttäjiä.

3.2 Vaaka

Vaakana työssä käytetään ADAM Equipmentin CPWplus 75 -vaakaa (kuva 5). Vaa'an maksimipaino on 75 kg ja tarkkuus 0.02 kg. Tuettuja painoyksikköjä ovat kilogramma (kg), punta (lb), unssi (Oz) ja punta-unssi (lb-oz), joten vaakaa pystyy käyttämään useammassa maassa. Vaa'assa on RS-232 (Recommended Standard 232) sarjaporttiliitäntä, jonka avulla vaakaa voi lukea tietokoneelta. Raspberry Pi -tietokoneessa ei sarjaporttiliitäntää ole, joten liitäntää varten käytettiin USB-adapteria, jolla kytkentä voitiin tehdä USB-porttiin. (7.)



Kuva 5. Työssä käytetty vaaka ja sen näyttömoduuli

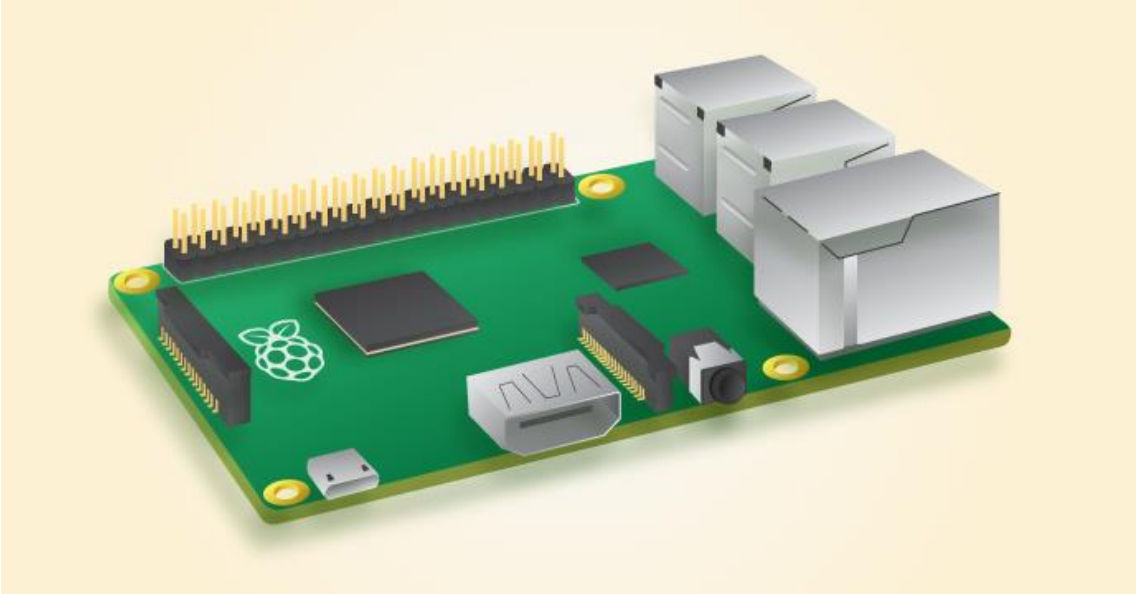
Vaa'assa on kaksi eri asetusta painon tulostamiseen. Ensimmäinen on jatkuva datan syöttö, jolloin vaaka lähettää noin neljä kertaa sekunnissa mittaamansa painon. Toinen vaihtoehto on lähettää paino näyttömoduulin print-painikkeen painalluksesta. Työssä päädyttiin kuitenkin käyttämään jatkuvaa datan siirtoa, koska sovelluksen käyttöliittymän painikkeesta pystyttiin vahvistamaan puntarin paino. Näin työssä ei tarvittu puntarin mukana tullutta näyttömoduulia.

3.3 Raspberry Pi

Raspberry Pi on yhden piirilevyn pienoistietokone, joka tukee useampaa eri käyttöjärjestelmää ja tarjoaa oman Raspbian-nimisen käyttöjärjestelmän. Työssä käytetty Raspberry Pi on Raspberry Pi 2 Model B. Tämän työn kannalta sen tärkeimpiä ominaisuuksia on:

- 900 Mhz ARM Cortex-A7 -neliydinprosessori

- 1 Gt RAM-muistia
- 4 USB-porttia
- Ethernet-portti
- Micro-SD-korttipaikka
- edullinen hinta
- pieni koko.



Kuva 6. Raspberry Pi 2 Model B (8)

Raspberry Pi tulee oletuksena ilman koteloa ja johtoja. Sille on kuitenkin saatavissa laaja valikoima erilaisia koteloita. Virtalähteenä käy tavallinen MicroUSB kännykän laturi, mutta on kuitenkin suositeltavaa, että virtalähteessä on vähintään 1,8 ampeerin virta (9).

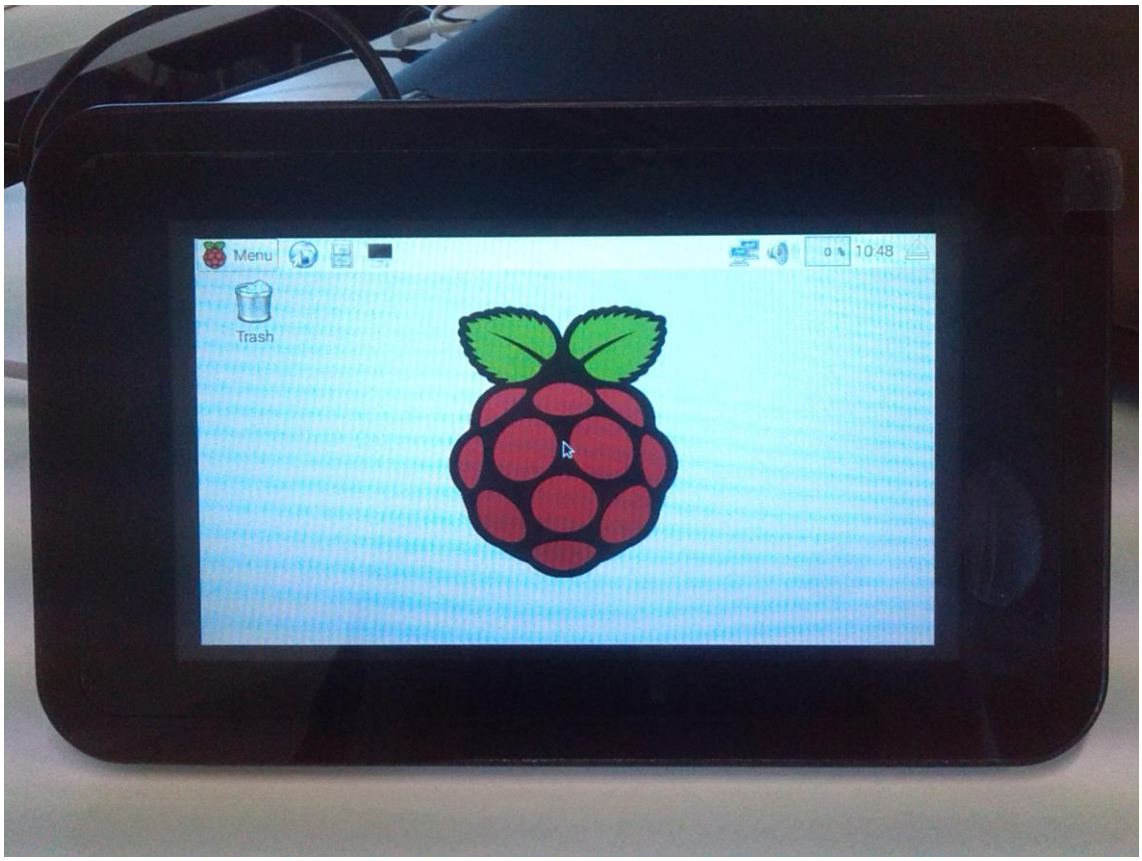
Raspberry Pi tukee muun muassa useita eri Linux-käyttöjärjestelmiä, sekä Windows IoT -käyttöjärjestelmää. Tähän insinöörityöhön valittiin Raspberry Pi -tietokoneelle kehitetty Raspbian-niminen käyttöjärjestelmä. Raspbian perustuu suosittuun Debian-käyttöjärjestelmään, mutta on optimoitu pienitehoisille tietokoneille sopivammaksi. Debianissa on laaja sovellusvalikoima, josta iso osa on myös Raspbianin käytettävissä. Tämä tekee Raspbianista erinomaisen valinnan työn alustaksi.

Käyttöjärjestelmän asennus toteutettiin käyttämällä Raspberry Pi -tietokoneen asennustyökalua NOOBS (New Out Of Box Software). NOOBS:lla voi asentaa useamman eri käyttöjärjestelmän helppokäyttöisestä käyttöliittymästä. Muistikortille voi asentaa samaan aikaan yhden tai useamman käyttöjärjestelmän. Jos käyttöjärjestelmiä asennetaan useampi, Raspberryn käynnistyksen yhteydessä valitaan mitä asennettua käyttöjärjestelmää käytetään. (10.)

3.4 Näyttö

Työn näyttönä käytettiin Raspberryn valmistajan omaa kosketusnäyttöä. Näytössä on 7 tuumaa ja 800x480 resoluutio. Näyttö tukee sormikosketusta, mikä on huomattava etu kynäohjattaviin kosketusnäyttöihin verrattuna. Näytössä tulee mukana oma kotelo, johon Raspberry Pi asennetaan. Näin se jää siististi piiloon näytön taakse. Kotelossa on aukot joista Raspberryn johdot saadaan kytkettyä. Kuvassa 7 nähdään näyttö käynnistetyllä Raspberry Pi -tietokoneella.

Kosketusnäyttö todettiin hyväksi ideaksi painon hyväksymisen kannalta. Kun klinikan asiakas punnitsee lemmikkiään, voi hän kosketusnäytöltä hyväksyä punnitsemisen tuloksen. Tähän kynäohjattava näyttö olisi ollut huomattavasti epäkäytännöllisempi asiakkaan käyttää. Kosketusnäytön ansiosta puntarin näyttömoduulia ei tarvittu, koska kosketusnäytöllä pystytään näyttämään punnitsemisen tulos, sekä tarvittavat painikkeet. Näin prosessista saatiin käyttäjäystävällisempi ja yksinkertaisempi.

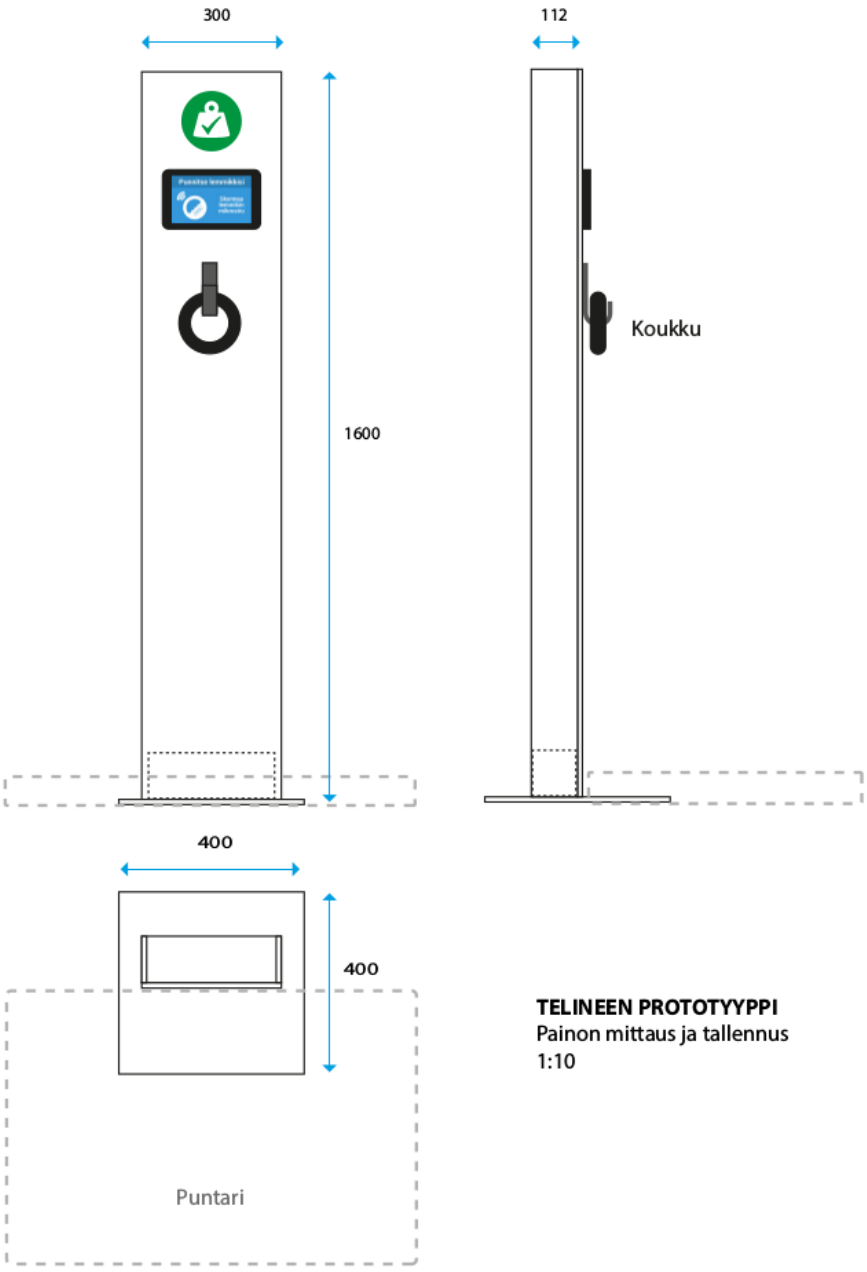


Kuva 7. Raspberry Pi -mikrotietokoneen kosketusnäyttö ja Raspbian-käyttöjärjestelmä

3.5 Kotelointi

Lemmikin punnitsemisessa näytettävä paino, sekä painon vahvistukseen ja nollaamiseen tarvittavat painikkeet löytyvät kosketusnäytöltä. Puntarin mukana tullutta näytön ja painikkeet sisältävää komponenttia ei tällöin tarvittu. Moduulia ei turhaan haluttu jättää näkyville tai eläinklinikoille piilotettavaksi, joten työssä päädyttiin koteloimaan kokonaisuus.

Kotelo suunniteltiin siten, että Raspberryn kosketusnäyttö upotetaan siihen ja puntarin näyttömoduuli piilotetaan kotelon sisälle. Tällä tavoin kokonaisuudesta saadaan huomattavasti siistimmän näköinen. Kuvassa 8 nähdään kotelon suunnitelmat. Puntari sijoitetaan kotelon eteen ja mikrosirunlukija ripustetaan koukusta koteloon, näytön alle. Kotelossa on jatkopistorasia, johon kytketään Raspberryn, sekä puntarin virtajohto. Näin kotelosta tarvitsee tuoda ulos vain kaksi johtoa, jatkopistorasian virtajohto, sekä verkkokaapeli.



Kuva 8. Projektin kotelointi FNS:n digi-AD Rurik Mahlbergin suunnittelemana

Kotelon materiaalina käytetään ensimmäisessä versiossa puuta. Mikäli painonpunnituspisteelle on tarpeeksi kysyntää, voidaan kotelon materiaali vaihtaa esimerkiksi metalliin. Materiaalin hankintaa ja leikkausta suunniteltiin tulevaisuudessa ulkoistettavaksi.

Projektin aikataulun puitteissa kotelointia ei ehditty toteuttaa, mutta materiaalit prototyypin rakennusta varten tilattiin.

4 Sovelluksen tekninen toteutus

4.1 Suunnittelu

Työssä otettiin mallia Unix-filosofiasta, jossa ohjelmista pyritään tekemään yksinkertaisia, pieniä, selkeitä, modulaarisia. Ohjelmat suunnitellaan tekemään yhtä selkeästi määriteltyä asiaa ja tekemään sen hyvin (11). Tämä tarkoittaa sitä, että Raspberry Pi ja laitteet toimivat mahdollisimman itsenäisesti ja järjestelmäriippumattomasti, sekä toteuttavat yhden toiminnon ja toteuttavat sen hyvin; painon tallennuksen eläimen mikrosirun perusteella. Yhteydet muihin järjestelmiin toteutetaan rajapinnan kautta, jolloin palvelimen toteutuksella, johon Raspberry ottaa yhteyden, ei ole merkitystä niin kauan kun sen toteuttama rajapinta on yhteensopivassa muodossa.

Ohjelman koodista tehtiin mahdollisimman yksinkertainen ja helppolukuinen. Tähän pyrittiin jakamalla ohjelman toimintoja pienempiin kokonaisuuksiin, kommentoimalla koodia missä tarpeen, sekä välttämällä ylimääräisiä toimintoja ja turhaa monimutkaisuutta. Tällä tavoin ohjelmaa on helppo ymmärtää, jos sitä tulevaisuudessa joutuu uusi henkilö ohjelmoimaan, tai ohjelman pariin palaa pitkän ajan jälkeen.

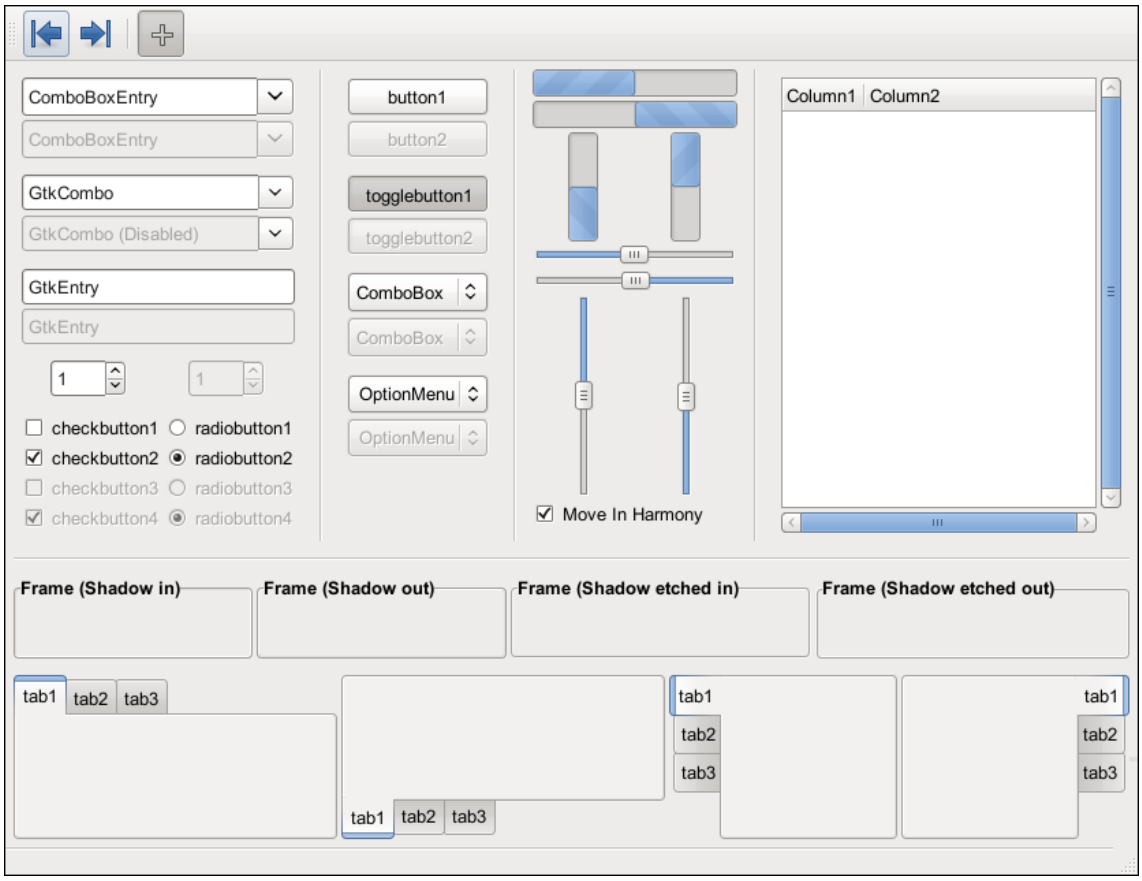
Toimintojen määrää pyrittiin myös pitämään mahdollisimman vähäisenä. Ohjelmistojen laatu ei aina parane uusien ominaisuuksien myötä ja ohjelma jossa on vähemmän ominaisuuksia, mutta on yksinkertaisempi käyttää, voi olla käyttäjälle miellyttävämpi vaihtoehto sen käytännöllisyyden, helpon lähestyttävyyden ja käytettävyyden takia (12). Uudet toiminnot lisäävät ohjelmistojen monimutkaisuutta niin käyttäjille kuin kehittäjille. Toimintojen lisäys tuo myös lisää kustannuksia toiminnon alustavan kehityksen ja ylläpidon muodossa. Epäsuoria kustannuksia tulee ohjelman paisumisesta, mikä lisää ohjelmoijan työtaakkaa ymmärtää mitä ohjelma tekee ja mistä sen eri toiminnot löytyvät. Raymond Eric S. kuvaa sovellusten monimutkaisuuden haittavaikutuksia seuraavasti:

> Unix-ohjelmoijan yksinkertaisuuden intohimon alla on pragmaattinen tosiasia: monimutkaisuus maksaa. Monimutkaista sovellusta on vaikeampi ajatella, vaikeampi testata, vaikeampi korjata, ja vaikeampi ylläpitää — ja ennen kaikkea, vaikeampi oppia ja käyttää. Monimutkaisuuden kustannukset, niin kovat kuin ne kehityksen aikana ovat, purevat pahiten julkaisun jälkeen. Monimutkaisuus luo vioille paikan pesiytyä, josta ne ilmaantuvat vaivaamaan maailmaa koko sovelluksen eliniän ajan. (13.)

Työssä harkittiin, pitäisikö mikrosiru ja painotieto lähettää, vaikka eläintä ei Provetin tietokannasta löydy. Tällöin Provet-järjestelmä voisi ottaa mikrosirun ja painon kuitenkin vastaan ja tarjota käyttäjilleen mahdollisuutta esimerkiksi liittää paino ja mikrosiru johonkin vanhaan tai uuteen eläimeen tietokannassa. Tämän toiminnon lisääminen työn sovellukseen ei olisi työmäärällisesti erityisen suuri, mutta Provet-ohjelmistoille se olisi suurempi, minkä lisäksi se lisäisi Provettiin monimutkaisuutta, tuomatta käyttäjälle merkittävää lisähyötyä. Tämän takia toiminto päätettiin alustavasti jättää toteuttamatta. Tarvittaessa toiminnon lisääminen on kuitenkin mahdollista vielä myöhemminkin.

4.2 Käyttöliittymä

Käyttöliittymän toteutuksessa oli tärkeä ottaa huomioon Python-tuki, Raspberryn heikko suorituskyky, sekä Raspbian-käyttöjärjestelmän tuki. Valinnassa päädyttiin GTK+-kirjastoon sen kattavan dokumentaation, pitkän iän ja Python-tuen takia, jolloin tukea olisi mahdollisimman helppo löytää. GTK+ on Linuxilla, Windowsilla ja Mac OSX käyttöjärjestelmillä toimiva kirjasto graafisten käyttöliittymien luomiseen. Kirjasto on kirjoitettu C-kielellä, mutta omaa liitännäiset useisiin eri ohjelmointikieliin, kuten Pythoniin. GTK+ on lisensoitu GNU LGPL -linsenssillä, mikä tarkoittaa, että se on ilmainen käytettäväksi myös kaupallisissa sovelluksissa. Tämän insinöörityön kannalta olennaisia ominaisuuksia kirjastossa olivat tekstin, kuvien ja painikkeiden lisääminen. (14.)



Kuva 9. GTK+-käyttöliittymäkirjaston elementtejä (15)

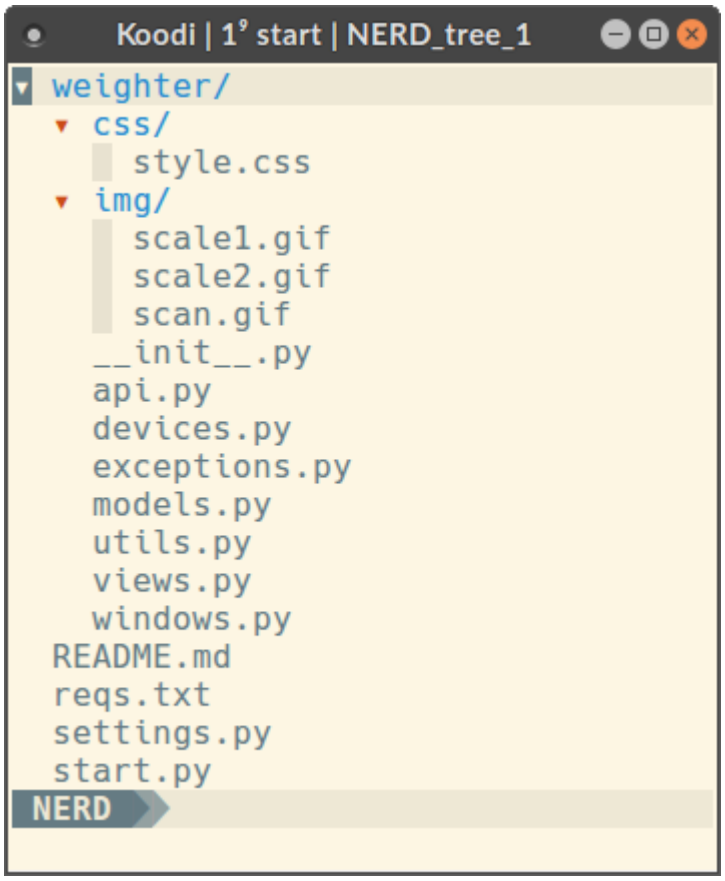
GTK+:ssa käyttöliittymään tulevat elementit ovat nimeltään widgettejä. Widgettien asettelu näytölle tapahtuu containereilla. Containerit ovat myös widgettejä, mutta niihin voidaan lisätä muita widgettejä, myös toisia containereita. Tässä työssä ainut käytetty container oli box, johon widgettejä voidaan lisätä vierekkäin tai allekkain. Ohjelman käyttöliittymän yksinkertaisuuden takia tämä oli tarkoitukseen riittävä. Kuvassa 9 nähdään GTK+:n erilaisia widgettejä.

GTK+ tukee CSS-määrittelyjä (Cascading Style Sheets) käyttöliittymän widgettien ulkonäön muokkaamiseen (16). CSS määrittää miten HTML-elementit (Hypertext Markup Language) näytetään ruudulla. Vaikka GTK+ ei käytä HTML-määrittelyä, tukee se silti CSS-syntaksin käyttöä widgettien tyylien muuttamiseen. Kirjasto ei tue kaikkia CSS-määrittelyjä, mutta tärkeimmät, kuten fontin, widgettien taustavärien muuttamisen, painikkeiden reunojen muokkaamisen ja sisennysten määrittelyn. GTK+ tukee muitakin tapoja ulkoasun määrittelyyn, mutta CSS sopi työhön sen yksinkertaisuuden takia.

Käyttöliittymän ulkoasun suunnitteli FNS:n digi-AD Rurik Mahlberg.

4.3 Projektin rakenne

Ohjelman logiikka jaettiin eri moduulien vastuulle. Tällä pyrittiin Unix-filosofian tyyliseen modulaarisuuteen, jossa ohjelman moduulit toteuttavat vain määrätyt ohjelman toiminnot. Nämä moduulit tarjoavat rajapinnan, jota kautta muu osa ohjelmasta käyttää moduulin toimintoja. Käytännössä työn moduulien rajapinnat olivat luokkien metodeja tai yksittäisiä funktioita.



Kuva 10. Projektin tiedostorakenne Vim-tekstieditorin The NERD Tree -pluginissa

Ohjelman moduulit, sekä CSS-tyylitiedosto ja kuvat asetettiin weighter-kansion alle. Tässä kansiossa on kaikki ohjelman logiikka. Projektin ylimmällä tasolla on ohjelman suoritukseen liittyvät tiedostot, kuten start.py-tiedosto, josta ohjelma käynnistetään.

Ohjelman käynnistämisestä ja asennuksesta kerrotaan lisää luvussa 6. Kuvassa 10 havainnollistetaan projektin tiedostojen jakoa.

4.3.1 Ikkuna

GTK+:n ikkuna-objekti piirtää ohjelman ikkunan. Ohjelman ikkuna määritellään windows-moduulissa. Ikkunalle voidaan määrittää onko se esimerkiksi koko näytön täyttävä, vai ikkunoitu ja tietyn kokoinen.

Työssä ikkunaa käytettiin myös ohjelman prosessin ja vaiheiden logiikan ohjaukseen käyttäen muiden moduulien toimintoja yhden toimivan kokonaisuuden rakentamiseksi. Ikkuna-moduulia voi täten kutsua ”päämoduuliksi”. Ikkunan vastuulla oli lukea laitteita käyttämällä devices-moduulia, näyttää oikea näkymä riippuen mitä ohjelmassa milläkin hetkellä tapahtui käyttäen views-moduulia, sekä hakea eläimen tiedot ja tallentaa paino models-moduulin avulla ja käynnistää prosessi alusta tarvittaessa.

4.3.2 Näkymä

Ohjelman näkymiä varten luotiin views-moduuliin view-luokkia, jotka periytyivät Box-widgetistä. Perimällä Box-widgetin luokka pystytään näkymään asettelemaan useita widgettejä, kuten tekstejä, painikkeita ja kuvia halutulla tavalla. Yksi view-luokka vastasi ohjelman yhden näkymän piirtämisestä. Näin grafiikan piirto saatiin eristettyä pieniin, helposti hallittaviin ja käytettäviin kokonaisuuksiin.

Näkymien vastuulle jätettiin pelkästään käyttöliittymän piirtäminen. Saattaa myös olla metodeja, joilla muutetaan kyseisen näkymän tekstejä. Koska näkymät eivät tiedä mitä muualla ohjelmassa tapahtuu, esimerkiksi punnitusnäkymässä näytettävä eläimen nimi ja paino tulivat aina ikkunalta.

Näkymän näyttäminen ikkunassa tapahtui kutsumalla halutun näkymän show_all-funktiota. Tämä funktio on GTK+ widgetin funktio, joka näyttää widgetin ja kaikki sen sisällä olevat widgetit. Widget määrittää myös hide-funktion, jolla nimensä mukaisesti näkymä saadaan aina halutessa piiloon. (17.)

4.3.3 Laitteet

Mikrosirunlukijalle ja puntarille tehtiin omat luokat, joilla on yhteinen kantaluokka Device devices-moduulissa. Laitteiden kantaluokka huolehti laiteyhteyden luonnin pyserialin avulla. Pyserial on python-moduuli, jolla pystyy helposti yhdistämään sarjaporttilaitteisiin, kirjoittamaan niihin ja lukemaan niistä (18). Device-kantaluokan perivissä Scale- ja Scanner-luokissa määritettiin laitteiden product ja vendor ID:t, joita kantaluokka käyttää yhteyden avaamiseen. Nämä ID:t yksilöllistävät nimensä perusteella laitteen ja valmistajan. Näin yhteinen yhdistämislogiikka saatiin jaettua yhdestä paikasta kantaluokan alustusmetodista kahdelle laitteelle.

Mikrosirunlukija ei toimi vielä pelkän pyserialin yhdistämisen jälkeen. Lukija vaatii yhdistämisen lisäksi kahden komennon kutsumisen, joiden jälkeen lukija siirtyy lukutilaan. Ensin lukijaa kutsutaan lähettämällä teksti ”PA\r”, johon lukija vastaa ”Z”. Toinen komento on ”RO\x01\r”, johon lukija vastaa ”HALO”. Komentojen jälkeen mikrosirunlukijalla voi lukea ja sen lukemat mikrosirujen numerot siirtyvät lukijan ulostulojonoon, josta ne ovat ohjelmalle luettavissa. Mikrosirunlukijan luokalle lisättiin funktio, jolla lukijan jonosta luetaan ja palautetaan arvoja. Mikäli arvoa ei löydy, tai se ei ole mikrosirun numero, funktio palauttaa tyhjää. Ohjelman ikkuna käyttää tätä luokan funktiota tarkastaakseen 250 millisekunnin välein, onko lukija löytänyt mikrosirua.

Puntarin alustaminen oli yksinkertaisempi ja siinä riitti pyserialilla yhteyden avaaminen. Puntarin luokalle luotiin myös oma metodin puntarin painon lukemiseen. Puntari palauttaa painon tekstin seassa, joten luokan lukumetodin vastuulla on parsia vastauksesta desimaaliluku ja palauttaa se. Metodi palauttaa aina puntarin ulostulosjonosta viimeisimmän painotiedon, johon puntari painotietoja lisää. Koska jonoon kertyy jatkuvasti dataa riippumatta ohjelman vaiheesta, on se tärkeä tyhjentää ennen puntarin lukua. Puntarin ulostulosjonossa painotieto on vanhimmasta uusimpaan, joten jos jonoa ei tyhjennä ennen lukua, voi sieltä tulla useita minuutteja vanhaa painotietoa.

4.3.4 Muut moduulit

Api-moduulin vastuulla on toteuttaa kaksi funktiota joilla käytetään Provet-järjestelmän rajapintaa. Ensimmäinen funktio ottaa vastaan mikrosirun numerosarjan ja palauttaa eläimen tiedot tai nostaa poikkeuksen, riippuen miten rajapinta vastaa. Toinen funktio

ottaa vastaan mikrosirun numeron, sekä painotiedon ja suorittaa painon tallennuksen. Rajapinnasta kerrotaan tarkemmin luvussa 5.

Exceptions-moduulissa määritellään kaikki ohjelman poikkeukset. Poikkeukset ovat nimensä mukaisesti poikkeavia tapahtumia ohjelmakoodissa. Poikkeukset voidaan Pythonissa ottaa kiinni try-except-toiminnolla. Ohjelman osa, josta epäillään poikkeuksia tulevan, ajetaan try-exceptin sisällä, jolloin voidaan määrittää mitä tietyn poikkeuksen tapauksessa tehdään. Työssä määritellään kolme eri poikkeusta: PatientNotFound, BackendError ja ConnectionError. PatientNotFound-poikkeus nostetaan, kun rajapinta ei löydä potilasta. Tämän poikkeuksen kohdalla ikkuna voi näyttää halutun näkymän ja palata ohjelmakierron alkuun. BackendError-tapauksissa ikkuna näyttää virheviestin ja palaa myöskin alkuun. BackendError voi olla esimerkiksi palvelimessa tapahtunut virhe, joista kerrotaan lisää kappaleessa 8. ConnectionError nostetaan, kun palvelimeen ei saatu yhteyttä. Tämä yleensä tarkoittaa asetuksissa olevan osoitteen olevan väärä.

```
try:
    # Hae potilaan tiedot Api-moduulin get_patient-funktiolla.
    patient_data = get_patient(microchip)
except PatientNotFound:
    # Näytä näkymä, jossa pyydetään ottamaan yhteys henkilökuntaan.
    self.views[”not_found”].show_all()
except (BackendError, ConnectionError):
    # Näytä virhenäkymä.
    self.views[”error”].show_all()
else:
    # Tiedot haettiin onnnistuneesti.
    # Näytä punnitusnäkymä.
    self.views[”scale”].show_all()
```

Koodiesimerkki 1: Esimerkki poikkeuksien käyttämisestä ohjelma logiikan ohjauksessa.

Utils-moduulissa on yleisiä muihin moduuleihin kuulumattomia apufunktioita tai -luokkia. Näillä voidaan muissa moduuleissa helpottaa tiettyjen toimintojen toteutusta.

Models-moduulissa määritellään luokka Patient. Patient-luokan alustukseen annetaan mikrosirun numero, jota luokka käyttää api-moduulin funktion kutsussa asettaakseen nimen, edellisen painon ja päivämäärän. Luokan vastuulla on myös toteuttaa painon tallentava metodi. Metodia kutsumalla ikkunasta annetulla painolla, käyttää luokka alustuksessa annettua mikrosirua ja metodikutsussa annettua painoa Api-moduulin painontallennuksen kutsumiseen.

4.3.5 Rakenteen yhteenveto

Aiemmin kuvatuilla moduuli- ja luokkajaoilla ohjelman logiikka ja vastuualueet saatiin jaettua helposti hallittaviin pienempiin kokonaisuuksiin. Näin kehittämistä ja virheiden korjaamista saatiin yksinkertaistettua. Jos esimerkiksi vain käyttöliittymän tietyn näkymän ulkoasua tarvitsi muuttaa, muutetaan sitä kyseisessä luokassa ja mikään muu ohjelman osio ei vaadi muutoksia, mikäli rajapinta ulospäin säilyy samana. Sama pätee muun muassa laitelukuihin. Jos puntarin lukemisessa tai siihen yhdistämisessä tarvitaan muutoksia tai korjauksia, voidaan puntarin luokkaa muuttaa ja muu järjestelmä, joka luokkaa käyttää pysyy samana.

Moduulien ansiosta ohjelmakoodista tulee myös huomattavan paljon luettavampaa. Hyvin nimetyillä ja selkeillä moduuleilla koodia lukemalla pystyy nopeasti päättelemään mitä kyseinen moduuli tekee. Ohjelmakoodissa pyrittiin yksinkertaisuuteen, jotta koodin ymmärtää nopeasti ja sitä on tarvittaessa helppo muuttaa. Mikäli koodia tarvitsee tulla vuosien kuluttua taas muokkaamaan, tai sitä muokkaa uusi henkilö, on selkeästi kirjotetulla ja moduuleihin jaetulla koodilla helppo aloittaa.

4.4 Kehitysmenetelmät

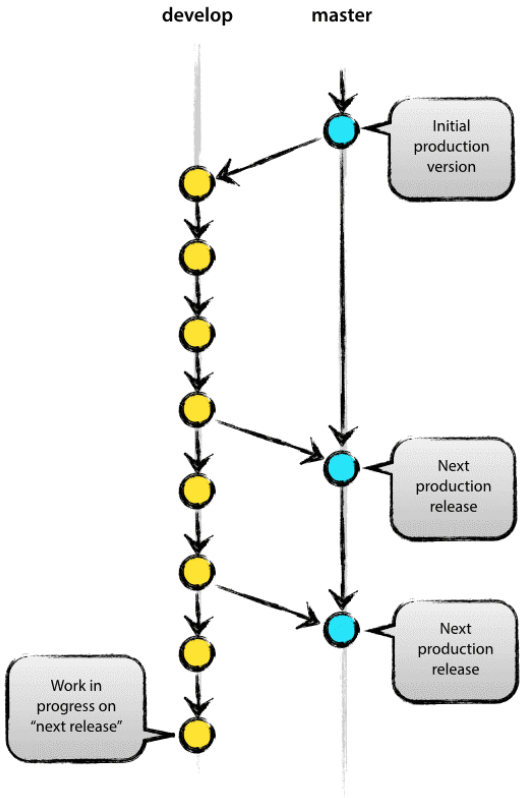
Vaikka työn sovelluspuoli toimii Raspberryllä, olisi sen kehittäminen Raspberryllä hyvin vaikeata ja epäkäytännöllistä. Tämän takia ohjelmointityö tehtiin pöytätietokoneella, josta muutokset siirrettiin Raspberrylle, aina kun ohjelmaa haluttiin Raspberryllä kokeilla. Tähän käytettiin apuna versionhallintajärjestelmää nimeltä Git.

4.4.1 Git-versionhallintajärjestelmä

Git on avoimen lähdekoodin hajautettu versionhallintajärjestelmä. Versionhallintajärjestelmä ylläpitää tiedostojen muutoksia, jolloin muutoksia voi helpommin palauttaa myöhemmin. Versionhallintaa käytetään usein ohjelmistojen koodin muutoksien ylläpidossa, mutta sitä voi käyttää melkein missä tahansa tiedostomuodossa. Versionhallintajärjestelmällä voi palauttaa tiedostoja edellisiin versioihin, vertailla versioiden muutoksia, sekä nähdä kuka on viimeksi tehnyt muutoksia. (19.)

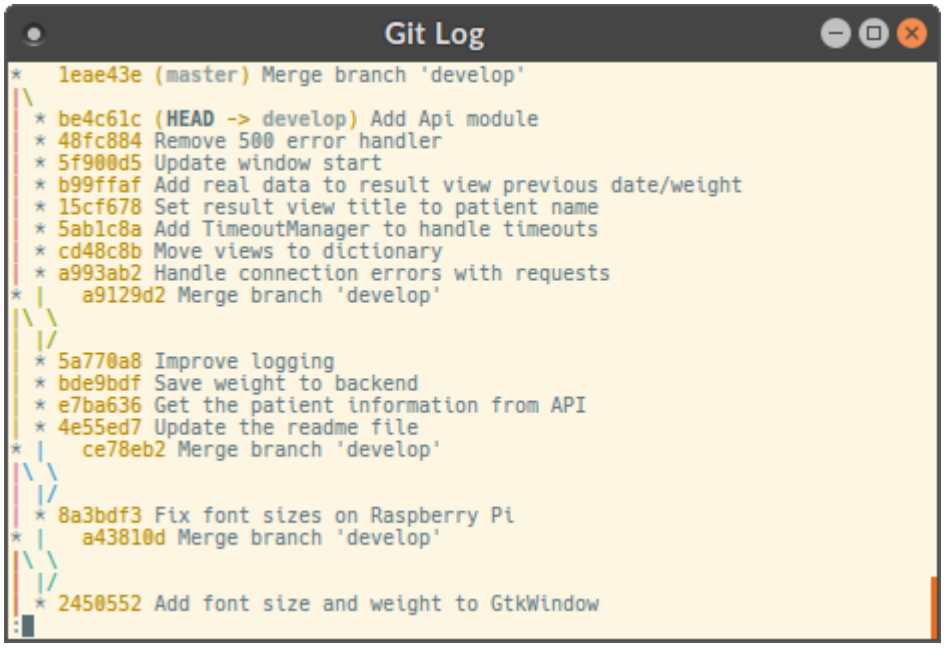
Gitin avulla projektien muutokset pystytään tallentamaan ja jakamaan helposti. Muutosten tallentaminen tapahtuu commit-toiminnolla, joka tallentaa projektin muuttuneet tiedostot ja asettaa muutoksille referenssin. Git ei tallenna tiedostoja, jotka eivät ole muuttuneet, vaan linkittää ne viimeisimpään muuttuneeseen versioon tiedostosta. Näin Git säästää huomattavan paljon levytilaa ja on nopea, kun muutokset pitää siirtää esimerkiksi toiselle koneelle. (19.)

Gitissä olennainen ominaisuus on haarat (branch). Haaroittamalla koodia järkevällä tavalla, on koodin hallinta huomattavasti helpompaa. Työssä käytettiin master- ja develop-haaroja. Master-haara vastaa aina tuotantokäytössä olevaa koodia. Tämän työn kannalta se tarkoittaa Raspberry Pi:llä olevaa koodia. Develop-haara sen sijaan sisältää aina viimeisimmät ohjelman muutokset, jotka ovat menossa tuotantoon. Kun Raspberry Pi halutaan päivittää viimeisimmillä muutoksilla, yhdistetään kaikki develop-haaran muutokset master-haaraan. (20).



Kuva 11. Koodin haarautumista develop- ja master-haaroihin Git-versionhallintajärjestelmässä (20)

Haarojen ansiosta on helppo seurata esimerkiksi mitkä muutokset ovat Raspberrylle vielä siirtämättä. Tämä nähdään tutkimalla mitä committeja on tullut develop-haaraan sen jälkeen, kun se on viimeksi yhdistetty master-haaraan. Gitissä on paljon ominaisuuksia, millä muutoshistoriaa voi tarkastella. Yksi on log-komento, joka listaa tallennetut muutokset.



Kuva 12. Projektin muutoshistoriaa git log -komennolla, jossa vasemmalla näkyy master- ja oikealla develop-haara

Commiteille täytyy aina antaa viesti. Viesteissä on otsikkorivi, sekä mahdollista sisältötekstiä. Kun muutoksien viestit ovat selkeitä ja kuvaavia, on esimerkiksi projektin muutoshistorian selaaminen tai muutosten peruminen huomattavan paljon helpompaa (21). Tämän vuoksi viestien kirjoitukseen kiinnitettiin erityistä huomiota.

4.4.2 Testaus Raspberry Pi -mikrotietokoneella

Ennen ohjelman testaamista Raspberryllä, täytyi siinä olleen projektin tiedostot päivittää lataamalla viimeisimmät muutokset kehityskoneelta. Vaikka kehityskoneessa olisi päivitetty master haara, ei tieto Raspberryyn siirry automaattisesti. Päivittämistä varten Raspberryyn yhdistettiin käyttäen SSH-yhteyttä (Secure Shell). Tämä mahdollistaa Raspberryn etäkäytön komentoriviltä jonka avulla projektin tiedostot päivitettiin lataa-

malla viimeisimmät muutokset kehitystietokoneen projektista. Toimenpide on nopea, koska Git siirtää vain muuttuneet tiedostot, eikä koko projektia.

Komentoriviltä SSH-yhteys ei mahdollista graafista piirtoa, jotta sovelluksen toimintaa voisi täysin testata. Tätä varten käytettiin Remmina-nimistä ohjelmaa. Remmina on vapaan lähden koodin etäohjausohjelma, joka tukee useampia yhteysprotokollia etäyhteyden muodostamiseksi (22). Remminalla yhdistettiin Raspberryyn, mikä mahdollistaa Raspberryn graafisen käytön. Ajettaessa sovellus näin, pystyttiin kehityskoneella kokeilemaan miten ohjelma toimii Raspberryllä.

Välillä ohjelma testattiin suoraan Raspberryllä ilman etäyhteyksiä. Tällä tavoin kosketusnäyttö voitiin todeta toimivaksi. Tämä vaati näppäimistön kytkemisen Raspberryyn, mikä tekee tavasta vaivalloisemman, kuin yhdistää Remminalla kehityskoneelta, jossa näppäimistö on jo kiinni.

5 Rajapinta Provet-ohjelmistoihin

Punnittavan eläimen tietojen haku ja painon tallennus tapahtuu HTTP-pohjaisen (Hypertext Transfer Protocol) rajapinnan välityksellä. HTTP-protokollaa käytetään verkkosivujen tiedonsiirrossa ja se sopii hyvin työn rajapinnan toteutukseen, koska Provet-ohjelmistot ovat selainpohjaisia ja käyttävät myös pääosin HTTP-protokollaa. HTTP toimii pyyntö-vastaus protokollana asiakkaan ja palvelimen välillä. Työssä Raspberry Pi on asiakas, joka tekee pyyntöjä palvelimeen, joka tässä tapauksessa käsittelee Provet Cloud- tai Provet Net-ohjelmiston tietoja. (23.)

5.1 HTTP-pyyntöjen tyypit

HTTP määrittelee joukon metodeja, jotka ilmaisevat mitä toimintoa halutaan toteuttaa. Tämän työn kannalta olennaisimpia ovat GET- ja POST-metodit. Muita metodeja ovat muun muassa DELETE, PUT, OPTIONS ja PUSH, mutta niitä ei tässä työssä käytetä. (24)

GET-pyynnöillä haetaan palvelimesta tietoa. Ne ovat ”turvallisia” siinä mielessä, että niiden ei ole koskaan tarkoitus tehdä muutoksia pyynnön kohteeseen. GET-pyyntöjä käytetään vain tiedon hakua varten (25). Asiakkaan tehdessä GET-pyynnön, palvelin palauttaa pyynnön kohteeseen liittyvät tiedot. GET-pyynnöissä voi olla parametrejä, joilla pyyntöä tarkennetaan. Tässä työssä pyyntö tehdään /get-patient-osoitteeseen ja parametrina annetaan mikrosirun numero, jolloin palvelin vastaa kyseisen mikrosirun omaavan eläimen tiedoilla, jos mikrosiruun liitetty eläin löytyy.

POST-pyynnöillä tallennetaan tietoa (25). Raspberry tekee POST-pyynnön palvelimelle osoitteeseen /save-weight. Tähän pyyntöön liitetään parametreiksi eläimen mikrosiru ja paino. Palvelin voi palauttaa POST-pyynnölle vastauksessa sisältöä, mutta tämä ei ole välttämätöntä. Usein sisältö voi olla muun muassa vahvistusviesti pyynnön onnistumisesta, osoite luodun tai muutetun sisällön tietoihin, tai kertaus tallennetuista tiedoista (26). Tässä työssä painon tallentavan POST-pyynnön ei oleteta palauttavan palvelimelta mitään sisältöä, mutta mikäli vastauksessa kuitenkin on sisältöä, sillä ei ole vaikutusta ohjelman suoritukseen.

Jatkossa tässä kappaleessa GET-pyynnöllä viitataan eläimen tiedon hakuun ja POST-pyynnöllä eläimen tiedon tallennukseen

5.2 HTTP-tilakoodit

Rajapinnassa hyödynnetään HTTP-tilakoodeja. Tilakoodit ovat kolmen numeron pituisia numeroita, joilla ilmaistaan onnistuiko asiakkaan lähettämän pyynnön prosessointi (27). Pyynnön vastauksessa on aina jokin koodi ja niitä on HTTP-standardissa määritelty paljon, mutta tässä työssä käytetyt ovat 200, 400 ja 404.

Onnistuneessa pyynnössä tulee vastauksena tilakoodi 200 (27). Tämä tarkoittaa GET-pyynnössä, että eläin on löytynyt mikrosirun perusteella ja eläimestä pyydetyt tiedot ovat vastauksen sisällössä. 200 tässä työssä viittaa kuitenkin vain siihen, että eläin on mikrosirun perusteella löytynyt ja palautuksessa on tiedot mitä eläimestä on löydetty ja ohjelman suoritusta voidaan jatkaa. 200 ei kuitenkaan takaa esimerkiksi sitä, että palautetuista tiedoista löytyy edes eläimen nimeä, jos sitä ei mikrosirun perusteella rekisteröidyn eläimen tietoihin ole tallennettu. POST-pyynnöissä 200 tarkoittaa, että paino on tallennettu onnistuneesti potilaan tietoihin, eikä vastauksen sisällössä palauteta mitään. Mikäli rajapinta onnistuneessa pyynnössä joka tapauksessa palauttaa jotakin, sitä ei huomioida.

Koodi 400 tarkoittaa, että palvelin ei ymmärtänyt pyyntöä (27). Tällöin vika on Raspberryn tekemässä pyynnössä. Ongelma voi esimerkiksi olla puuttuva mikrosirun numero, tai POST-pyynnöstä puuttuva paino. Tämän tilakoodin ei pitäisi normaalitapauksissa tulla, koska Raspberryn koodi ja asetukset määritetään siten, että pyyntö on oikein. 400 voi kuitenkin palautua esimerkiksi jos palvelin odottaa erinimistä parametriä, kuin mitä Raspberry lähettää, (esimerkiksi parametriä microchip_number parametrin microchip sijaan). Tällöin vika on palvelimen toteutuksessa ja se on muutettava vastaamaan Raspberryn pyyntöä. Raspberry määrittää aina oikean pyynnön muodon ja rajapinnan toteuttavien palvelimien on mukauduttava siihen.

Mikäli eläintä ei löydy, tulee rajapinnan palauttaa tilakoodi 404. Tämä tarkoittaa HTTP-tilakoodeissa sitä, että pyydettyä resurssia ei löydy (27). Tämän palautetun tilakoodin avulla pystytään ohjelmassa pyytämään asiakasta ottamaan henkilökuntaan yhteyttä,

koska sovellus ei tue uuden potilaan lisäystä Raspberryn kautta. Sekä GET- että POST-pyynnöt tarvitsevat mikrosirun numeron lemmikin yksilöimiseen ja voivat täten palauttaa 404, jos lemmikkiä ei löydy. 404-vastauksissa ei tarvita sisältöä, koska tilakoodi tarkoittaa yksiselitteisesti sitä, että pyydettyä resurssia ei löydetty.

Mikäli palvelin palauttaa minkä tahansa muun tilakoodin kuin 200, 400, tai 404, kirjataan tilakoodi ja vastauksen sisältö lokiin. Yksi näistä voi olla tilakoodi 500, joka tarkoittaa palvelimen sisäistä, asiakkaan pyynnöstä riippumatonta virhettä.

Virheessä 400 rajapinta voi palauttaa virheviestin. Tätä etsitään error-muuttujasta. Virheviestiä ei näytetä asiakkaille, mutta se lähetetään sähköpostilla punnituspisteen tekniseen tukeen. Sähköpostiin liitetään virheviestin lisäksi Raspberry Pi:n MAC-osoite, sekä asetukset. Näin virhettä voidaan helposti alkaa selvittämään etänä. Tilakoodin 400 vastaukset ovat ainoita vastauksia joista etsitään error-muuttujaa. Kaikista muista tilakoodeista, kuten 500:sta sen oletetaan puuttuvan. Tällöin lähetetään vastauksen sisältö kokonaisuudessaan tilakoodin kanssa tekniseen tukeen.

5.3 Rajapinnan väärinkäytökset

Rajapintaa ei voida pitää POST-pyynnöille täysin avoimena. Tällöin kuka tahansa, joka tietää osoitteen, millä rajapintaan yhdistetään, sekä muuttujat, joissa rajapinta ottaa tietoa vastaan, voisi tallentaa äärettömän määrän painotietoja satunnaisille potilaille.

Jotta väärinkäytöksiä voitaisiin minimoida, työssä toteutetaan aluksi yksinkertaisempi salasanasuojaus, jotta järjestelmä saadaan testaukseen. Kun toteutusta on testattu eläinklinikalla ja asiakkaat osoittavat punnituspisteeseen kiinnostusta, otetaan tietoturva uudestaan tarkempaan tarkasteluun ja sitä kehitetään tarpeiden mukaan.

Työn aikataulun puutteissa salasanasuojausta ei ehditty työhön toteuttaa tai tarkemmin kuvata.

5.4 Testirajapinta

Työhön tehtiin muusta projektista erillinen testirajapinta, joka palauttaa tietoa, jotta Raspberryn rajapintakutsuja voidaan kokeilla ja josta tuotantorajapintojen ohjelmoijat voivat ottaa mallia. Rajapinta toteuttaa vaaditut käyttötapaukset mikrosirun perusteella eläimen tiedon hakemiseen, sekä painon tallentamiseen. Palautettu sisältö on käsin syötettyä ja keksittyä data.

Testirajapinnan tekninen toteutus tehtiin käyttäen PHP-ohjelmointikieltä (PHP: Hypertext Preprocessor). Tämä siksi, että PHP on yksinkertainen ja nopea ottaa käyttöön. Rajapinta on myös niin yksinkertainen, ettei testitapaukseen tarvinnut käyttää erityisen paljon aikaa ja vaivaa.

Rajapinnan luonnissa käytettiin apuna Lumen-nimistä web-sovelluskehystä. Lumen on erittäin nopea mikro-sovelluskehys, joka perustuu laajempaan Laravel sovelluskehykseen. Sovelluskehyksien etuina on niiden tarjoamat työkalut, jotka nopeuttavat ohjelman kehitystä. Lumenissa työkaluja on huomattavasti vähemmän kuin Laravelissa, mutta tarpeeksi testirajapinnan laajuuteen nähden. (28.)

Yksi olennainen ominaisuus Lumenissa on verkkosivun osoitteen perusteella oikeiden funktioiden kutsu (29). Osoitteiksi määritellään get-patient ja save-weight, jotka kutsuvat asianmukaisia funktioita, joista löytyy logiikka pyyntöjen prosessointiin. Tällöin menemällä osoitteeseen http://testirajapinnanosoite/get-patient, palautuu eläimen tiedot. Lumen tarjoaa myös apuvälineet HTTP-palautteen luomiseksi (30). Vastauksen tiedot palautetaan JSON-muodossa, joka on yksinkertainen ja suosittu tiedon esitysmuoto verkkorajapinnoissa (31).

Lumen mahdollistaa myös yksikkötestien luonnin (32). Yksikkötestit ovat funktioita, jotka suorittavat tietyn ohjelman toiminnon ja tarkastelevat sen tulosta. Testien avulla varmistutaan, että koodi toimii halutulla tavalla. Mikäli koodia muokataan tulevaisuudessa, voidaan testit aina ajaa uudestaan, jotta varmistutaan testattujen toimintojen toimivan edelleen samalla tavalla kuin ne toimivat testejä kirjoittaessa. Testit vähentävät koodista löytyviä virheitä, sekä helpottavat koodin rakenteen ymmärtämistä. (33.)

Yksikkötestit ajetaan komentoriviltä, jonka jälkeen komentorivillä näkyy testien tulokset. Tuloksissa näkyy kuinka monessa testissä tarkastukset menivät läpi, kuinka monessa ei ja kuinka monessa testissä tapahtui virhe. Testirajapintaan luotiin yksikkötestit molempiin käyttötapauksiin eri parametrien yhdistelmillä. Esimerkiksi yksi testi kokeilee kutsua rajapinnan potilaan tiedon hakua ilman mikrosirun numeroa. Testissä tarkastetaan, että rajapinta palauttaa tilakoodin 400 ja viestin puuttuvasta mikrosirusta. Toisessa testissä annetaan mikrosiru, jolla ei potilasta löydy. Kyseisessä testissä varmistetaan, että rajapinta palauttaa tilakoodin 404. Kuvassa 13 nähdään erilaisten testien pyyntöjä, sekä odotettuja vastauksia.

Backend | 1³ test | tests/GetPatientTest.php

```php
class GetPatientTest extends TestCase
{
    public function testMethodPost()
    {
        $this->post("/get-patient/");
        $this->assertEquals(405, $this->response->status());
    }

    public function testWithoutMicrochip()
    {
        $this->get("/get-patient/");
        $this->assertEquals(400, $this->response->status());
        $this->seeJsonEquals(["error" => "Missing parameter microchip."]);
    }

    public function testUnregisteredMicrochip()
    {
        $response = $this->call("GET", "/get-patient/", ["microchip" => "99999"]);
        $this->assertEquals(404, $response->status());
        $this->seeJsonEquals([]);
    }

    public function testPatientWithoutPreviousWeight()
    {
        $response = $this->call("GET", "/get-patient/", ["microchip" => "54321"]);
        $this->assertEquals(200, $response->status());
        $this->seeJsonEquals([
            "name" => "Musse",
            "previous_date" => "",
            "previous_weight" => "",
        ]);
    }

    public function testPatientWithPreviousWeight()
    {
        $response = $this->call("GET", "/get-patient/", ["microchip" => "12345"]);
        $this->assertEquals(200, $response->status());
        $this->seeJsonEquals([
            "name" => "Jeppe",
            "previous_date" => "23.04.2016",
            "previous_weight" => "12 kg",
        ]);
    }
}
```

NORMAL master tests/GetPatientTest.php 10% 5:1

Kuva 13. Testirajapinnan eläimen tiedon haun testejä

Sen lisäksi, että testit pitävät huolen, että rajapinta toimii oikein esimerkiksi mahdollisten tulevien teknisten muutosten jälkeen, ne toimivat samalla koodin dokumentaationa (34). Testejä lukemalla on helppo ymmärtää, miten rajapinnan kuuluu toimia. Testeissä määritellään eri skenaariot, mihin rajapinnan tulee varautua ja miten niihin kuuluu vastata. Näin Provet-ohjelmistoihin rajapintoja luodessa on testirajapinnan testeistä helppo katsoa, että vastaavat tapaukset on toteutettu myös tuotantopuolella ja vastaukset ovat identtiset.

6 Järjestelmä tuotantokäytössä

Kun Raspberry Pi sekä oheislaitteet on lähetetty asiakkaalle käytettäväksi, täytyy sovellusta pystyä tarvittaessa päivittämään etänä. Tätä varten käyttöön otettavaksi suunniteltiin Finnish Net Solutionsilla kehitetty BlackBox-järjestelmä, jota yrityksessä käytetään muissakin Raspberry Pi -pohjaisissa sovelluksissa.

Työn aikatauluun ei saatu mahdutettua BlackBox-järjestelmän käyttöä ja testausta, mutta tässä kappaleessa käydään läpi miten järjestelmä toimii ja miten sitä tullaan tulevaisuudessa käyttämään projektissa hyödyksi.

6.1 BlackBox-järjestelmä

Verkkoselainpohjaisessa BlackBox-hallintajärjestelmässä ylläpidetään asiakkailla käytössä olevien Raspberry Pi -tietokoneiden asetuksia. Ideana on, että Raspberry lataa aina käynnistyessään BlackBox-hallintajärjestelmästä omat asetuksensa. Asetuksissa on myös linkki Zip-tiedostoon, jonka Raspberry lataa ja purkaa. Puretuissa tiedostoissa täytyy olla start.py tiedosto, jonka Raspberry ajaa Pythonilla. Start-tiedosto voi tehdä kaiken tarvittavan Raspberryllä halutun ohjelman ajamiseksi. Tiedostossa voidaan esimerkiksi asentaa tarvittavia ohjelmistoja, määrittää käyttöjärjestelmän asetuksia ja lopuksi käynnistää itse ohjelma. Kaikki BlackBox järjestelmässä olevat Raspberry Pi -mikrotietokoneet ovat tässä mielessä identtisiä.

Järjestelmään täytyy ensin määritellä rooleja. Roolille liitetään Zip-tiedosto, josta Raspberry saa ohjelmakoodinsa. Roolille voidaan myös määrittää halutun määrän Raspberry-kohtaisia asetuksia. Yhtä roolia voi käyttää rajaton määrä Raspberry Pi -mikrotietokoneita.

Jotta Raspberry Pi pystyy lataamaan BlackBox-järjestelmästä oikeat asetukset ja ohjelmakoodin, täytyy järjestelmässä yhdistää Raspberryn MAC-osoitteeseen (Media Access Control) oikea rooli. MAC-osoitteella Raspberry Pi pystytään yksilöimään, koska MAC-osoite on aina laitekohtainen (35). Kun Raspberry liitetään rooliin, voidaan samalla määrittää halutut asetukset, jotka kyseinen Raspberry saa käynnistyessään. Tässä työssä asetuksiin tulisi rajapinnan osoite, jota ohjelmassa käytetään. Osoite on

eri jokaisella Raspberryllä, ellei yhdellä eläinklinikalla ole kahta eläimen punnituspistettä.

Joskus voi olla tarpeen ottaa Raspberry Pi asiakkaalta pois käytöstä. Syynä voi olla esimerkiksi väärinkäytökset, asiakkaan maksamaton lasku, tai sopimuksen purkaminen. BlackBox-järjestelmästä voidaan tällöin yksinkertaisesti poistaa asiakkaalle lähetetty Raspberry Pi MAC-osoitteen perusteella, jonka jälkeen laite ei enää pysty lataamaan asetuksiaan ja ohjelmakoodia.

Jos sovellukseen täytyy tehdä päivityksiä, ladataan BlackBox-järjestelmään uusi Zip-tiedosto, jossa on päivitetty versio ohjelmasta. Raspberryt seuraavassa käynnistyksessä lataavat aina uusimman version Zip-tiedostosta. Koska Raspberryt voivat olla useita päiviä käynnissä eläinklinikalla, lisättiin niihin toiminto, mikä käynnistää Raspberryn aina uudestaan keskiyöllä. Tällä tavoin päivitykset latautuvat vähintään kerran vuorokaudessa.

6.2 Virhetilanteet ja laitteiden vikaantuminen

Mikäli ohjelmaan tulee jokin virhe, voidaan Raspberry Pi käynnistää uudelleen irrottamalla virtajohto ja kytkemällä se takaisin. Raspberry käynnistyy heti itsestään, kun se saa virtaa ja lataa asetukset, ohjelmakoodit ja käynnistää sovelluksen. Tämän pitäisi korjata useimmat virhetilanteet, joita ei ohjelmassa ole välttämättä osattu ottaa huomioon.

Jos laitteen uudelleenkäynnistys ei auta, voidaan asiakkaalle tarvittaessa lähettää kokonaan uusi Raspberry Pi. Ongelmia uuden lähettämisen suhteen aiheuttaa kotelointi. Raspberry on kosketusnäytön takana olevassa kotelossa, mistä Raspberry on asiakkaalle vaikea vaihtaa. Itse kosketusnäyttökin upotetaan koteloon, jolloin kosketusnäyttökin täytyy osata vaihtotilanteissa ottaa pois ja laittaa takaisin paikalleen.

On myös mahdollista, että mikrosirunlukija tai puntari lakkaa toimimasta. Tällöin asiakkaalle voidaan myös lähettää uusi laite. Laitteen vaihtoon täytyy laatia tarkat ohjeet, jotta asiakas pystyy sen itse tekemään. Mikrosirunlukijan ja puntarin kohdalla tämä on

kuitenkin helpompaa, koska riittää, että vanhan laitteen irrottaa Raspberrystä ja uuden liittää tilalle.

Suomen sisällä Raspberry voidaan kohtuullisen matkan päässä käydä tarvittaessa vaihtamassa FNS:n puolesta, jos asiakas ei itse siihen pysty. Järjestelmää on kuitenkin tarkoitus lähettää myös ulkomaille, jolloin paikan päälle lähteminen ei ole järkevää. Tätä varten viallisten komponenttien vaihtoa varten täytyy tulevaisuudessa toteuttaa erittäin yksityiskohtaiset ja tarkat ohjeet.

7 Yhteenveto

Työn tavoitteena oli luoda Finnish Net Solutionsille eläimen painonpunnituspiste, joka tallentaa eläimen mikrosirun perusteella eläimen painon Provet-ohjelmistoon. Työn teknisessä toteutuksessa päästiin onnistuneesti sille määritettyihin tavoitteisiin. Järjestelmällä pystyy tunnistamaan ja punnitsemaan asiakkaan eläimen ja lähettämään painotiedon eteenpäin Provet-järjestelmään. Provet-ohjelmistoihin tulevia rajapintoja varten saatiin luotua testirajapinta, jolla Raspberry Pi -tietokoneen toimintaa pystyttiin testaamaan ja josta tuotantorajapintojen ohjelmoijat voivat ottaa mallia. Käyttöliittymästä ja prosessista onnistuttiin luomaan yksinkertainen ja selkeä, jolloin järjestelmää ennenkin käyttämätön asiakas pystyy sen toimintaperiaatteen nopeasti ymmärtämään ilman erillistä ohjausta.

Projektin aikataulun puitteissa ei BlackBox-järjestelmää ehditty ottaa käyttöön. Tämän takia sovellusta ei pystytä päivittämään etänä, eikä se tällöin vielä ole tuotantokelpoinen. Ohjelmassa todettiin myös puuttuvan toiminto mikrosirun lukemisen ohittamiseen. Eläimillä ei aina ole mikrosirua, jolloin mikrosirun lukeminen täytyy pystyä ohittamaan. Tätä ei työhön ehditty lisätä, mutta ongelma suunniteltiin korjattavaksi lisäämällä ensimmäiseen prosessin vaiheeseen painike, jolla mikrosirun luku ohitetaan ja asiakas pääsee punnitsemaan lemmikkinsä.

Ohjelman teknisessä toteutuksessa onnistuttiin seuraamaan Unix-filosofiaa hyvin. Ylimääräisiä toimintoja ei lisätty, ohjelma pysyi pienenä, modulaarisen, sekä se hoitaa määritetyt tehtävänsä hyvin. Ohjelma on helposti luettavissa ja ymmärrettävissä, jolloin sitä on vuosienkin päästä helppo tarvittaessa korjata, vaikka koodia lukisi ensimmäistä kertaa.

Mikäli Provet-tuoteperheeseen tulee uusia ohjelmia, tai johonkin nykyiseen Provet-ohjelmaan halutaan punnituspiste lisätä, on työ niihin helposti liitettävissä. Tällöin painotiedot vastaanottavalle palvelimelle luodaan vastaava rajapinta, mikä työssä on kuvattu. Koska rajapinta toimii HTTP-protokollan kautta, on työ hieman helpompi liittää selainpohjaisiin ohjelmistoihin, mutta liitos on täysin mahdollinen myös muun tyyppisiin ohjelmistoihin.

Työn lopputuloksen on tarkoitus olla käytettävissä myös muissa maissa, kuin Suomessa. Käännöksiä ei ehditty toteuttamaan, mutta ne tullaan lisäämään, kun järjestelmä on ensin saatu testattua Suomessa. Myös painoyksiköiden huomiointi jäi puuttumaan. Tämä suunniteltiin tulevaisuudessa toteutettavan BlackBox-järjestelmän asetuksena. Kun painoyksiköiden määrittämisen tuki lisätään, täytyy se ottaa huomioon rajapinnan toteutuksessa, jotta palvelin ja Raspberry Pi käyttävät samaa yksikköä.

Punnituspisteelle suunniteltiin jo kahta maksutapamallia. Ensin mietittiin eläinklinikoilta perittävän yhtä maksua, jonka jälkeen laitteet lähetetään asiakkaalle. Tässä mallissa tosin ongelmalliseksi todettiin määrittää kuinka kauaksi aikaa FNS sitoutuu ylläpitämään punnituspisteen tukea. Tämän takia todennäköisempänä maksutapamallina pidetään kuukausimaksullista palvelua. Asiakas maksaa sovitun summan kuukausittain FNS:n toimittamista laitteista ja niiden ylläpidosta. Kuukausimaksullisessa mallissa hyötyinä ovat suuremmat tulot pidemmällä aikavälillä, sekä tuen lopettaminen sopimuksen irtisanomisen jälkeen.

Työlle on jo löydetty testikäyttäjiä ja sille on osoitettu kiinnostusta muun muassa eräästä ruotsalaisesta eläinklinikasta. Projektin kehitystä jatketaan.

Lähteet

1 Green, Don. 2015. The Importance of Simplicity in User Interface (UI) Design. Verkkojulkaisu. Einsights. <http://einsights.com/the-importance-of-simplicity-in-user-interface-ui-design/>. 26.10.2015. Luettu 4.5.2016.

2 Seebach, Peter. 2001. The cranky user: The Principle of Least Astonishment. Verkkojulkaisu. IBM. < http://www.ibm.com/developerworks/web/library/us-cranky10/index.html>. 1.8.2001. Luettu 4.5.2016.

3 Halo Microchip Scanner. Verkkojulkaisu. Scanner Angel. <http://scannerangel.com/en/halospec.html>. Luettu 12.4.2016.

4 Compulsory Microchipping for Dog Owners. Verkkojulkaisu. Petlog. <https://www.petlog.org.uk/pet-owner/compulsory-microchipping-for-dog-owners/>. Luettu 12.4.2016.

5 Sweden Pet Passport & Regulations. Verkkojulkaisu. PetTravel. <http://www.pettravel.com/immigration/Sweden.cfm>. Luettu 12.4.2016.

6 Pet Microchips for Dogs and Cats. Verkkojulkaisu. <http://www.pettravel.com/0001225.cfm>. Luettu 12.4.2016.

7 ADAM CPWPlus 75 Specifications. Verkkojulkaisu. ADAM Equipment. <http://www.adamequipment.com/am/details/122/96>. Luettu 12.4.2016.

8 Raspberry Pi 2 Model B. Verkkojulkaisu. Raspberry Pi. <https://www.raspberrypi.org/products/raspberry-pi-2-model-b/>. Luettu 20.4.2016.

9 Frequently Asked Questions. Verkkojulkaisu. Raspberry Pi. <https://www.raspberrypi.org/help/faqs/>. Luettu 4.5.2016.

10 NOOBS Setup. Verkkojulkaisu. Raspberry Pi. <https://www.raspberrypi.org/help/noobs-setup/>. Luettu 4.5.2016.

11 Raymond, Eric. 2003. Basics of the Unix Philosophy. Verkkojulkaisu. <http://www.catb.org/esr/writings/taoup/html/ch01s06.html>. Luettu 2.5.2016.

12 Gabriel, Richard. The Rise of ”Worse is Better”. Verkkojulkaisu. <https://www.jwz.org/doc/worse-is-better.html>. Luettu 2.5.2016

13 Raymond, Eric. 2003. Speaking of Complexity. Verkkojulkaisu. <http://www.catb.org/esr/writings/taoup/html/ch13s01.html>. Luettu 2.5.2016.

14 GTK+ kotisivut. Verkkojulkaisu. <http://www.gtk.org/>. Luettu 20.4.2016.

15 GTK+ features. Verkkojulkaisu. <http://www.gtk.org/features.php>. Luettu 20.4.2016.

16 Garnacho, Carlos. 2015. Styling GTK+ with CSS. Verkkojulkaisu. <https://thegnomejournal.wordpress.com/2011/03/15/styling-gtk-with-css/>. 15.3.2015. Luettu 20.4.2016.

17 GtkWidget reference. Verkkojulkaisu. GTK+. <https://developer.gnome.org/gtk3/stable/GtkWidget.html>. Luettu 20.4.2016.

18 Pyserial documentation. Verkkojulkaisu. <https://pythonhosted.org/pyserial/pyserial.html>. Luettu 25.4.2016.

19 Git Book, Basics. Verkkojulkaisu. <https://git-scm.com/book/en/v2/Getting-Started-Git-Basics>. Luettu 29.4.2016

20 Driessen, Vincent. 2010. A successful Git branching model. Verkkojulkaisu. <http://nvie.com/posts/a-successful-git-branching-model/>. Luettu 17.4.2016.

21 Beams, Chros. 2014. How to Write a Git Commit Message. Verkkojulkaisu. <http://chris.beams.io/posts/git-commit/>. Luettu 17.4.2016

22 Remmina kotisivut. Verkkojulkaisu. <http://www.remmina.org/wp/>. Luettu 17.4.2016.

23 Beal, Vangie. HTTP – HyperText Transfer Protocol. Verkkojulkaisu. Webopedia. <http://www.webopedia.com/TERM/H/HTTP.html>. Luettu 2.5.2016.

24 RFC 2616. Verkkojulkaisu. W3. <https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html >. Luettu 2.5.2016

25 Korpela, Jukka. 28.9.2003. Methods GET and POST in HTML forms - What's the difference? Verkkojulkaisu. Tampereen Teknillinen Yliopisto. <http://www.cs.tut.fi/~jkorpela/forms/methods.html>. Luettu 4.5.2016.

26 Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. 2014. RFC 7231. <http://tools.ietf.org/html/rfc7231#section-4.3.3>. 6.2014. Luettu 4.5.2016.

27 HTTP response codes. Verkkojulkaisu. Mozilla Developer Network. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Response_codes. Luettu 4.5.2016.

28 Lumen. Verkkojulkaisu. <https://lumen.laravel.com/>. Luettu 4.5.2016.

29 Lumen documentation, HTTP Routing. Verkkojulkaisu. <https://lumen.laravel.com/docs/5.2/routing>. Luettu 4.5.2016.

30 Lumen documentation, http Responses. Verkkojulkaisu. <https://lumen.laravel.com/docs/5.2/responses>. Luettu 4.5.2016.

31 Will. 2011. Verkkojulkaisu. mLab blog. <http://blog.mlab.com/2011/03/why-is-json-so-popular-developers-want-out-of-the-syntax-business/>. 30.3.2011. Luettu 4.5.2016.

32 Lumen documentation, Testing. Verkkojulkaisu. <https://lumen.laravel.com/docs/5.2/testing>. Luettu 4.5.2016.

33 Kettunen, Joni. 2015. Miksi kirjoittaa yksikkötestejä? Verkkojulkaisu. <https://kultainenkoodi.wordpress.com/tag/yksikkotestaus/>. 31.1.2015. Luettu 4.5.2016.

34 Millkin, Ann. 2014. The Benefits of Unit Testing. Verkkojulkaisu. Seque Technologies. <http://www.seguetech.com/blog/2014/10/10/benefits-unit-testing>. 10.10.2014. Luettu 4.5.2016.

35 What is a MAC address? Verkkojulkaisu. What Is My IP Address. <http://whatismyipaddress.com/mac-address>. Luettu 4.5.2016.