Jeremikael Juhala

KODIN SÄHKÖNKULUTUKSEN ETÄSEURANTAJÄRJESTELMÄ

Tietotekniikan koulutusohjelma

2017

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

KODIN SÄHKÖNKULUTUKSEN ETÄSEURANTAJÄRJESTELMÄ

Juhala, Jeremikael
Satakunnan ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Joulukuu 2017
Ohjaaja: Ylikoski, Mauri
Sivumäärä: 25
Liitteitä: 2

Asiasanat: etäseuranta, Raspberry Pi, Ruby

Työn tarkoituksena oli rakentaa järjestelmä, joka lukee kodin sähkömittarilta kulutustietoja, ja josta käyttäjä pääsee näkemään ne helposti. Vaatimuksia työlle olivat tiedon reaaliaikaisuus, mahdollisuus seurata kulutusta lyhyellä aikavälillä, sekä mahdollisuus seurata kulutusta mistä tahansa.

Laitteistoksi työhön valittiin Raspberry Pi tietokone. Tähän valintaan päädyttiin, ettei työssä tarvitsisi käyttää useampaa laitetta, vaan kaiken saisi tehtyä yhdellä laitteella. Kulutustietojen lukemiseen sähkömittarilta hyödynnettiin mittarin lediä. Ledin lukemiseen käytettiin valoherkkää vastusta. Luenta päätettiin suorittaa Python-ohjelmointikielellä tehdyllä luentaohjelmalla. Luennan tuottamat tiedot tallennettiin tietokantaan usein, jotta saatiin toteutettua tiedon reaaliaikaisuus ja esitys lyhyellä aikavälillä.

Mahdollisuus seurata kulutusta mistä tahansa toteutettiin tekemällä Web-sovellus, jolla käyttäjä voi seurata tietoja. Web-sovelluksen toteutukseen käytettiin Ruby on Rails ohjelmistokehystä ja MVC-arkkitehtuuria. Lisäksi Web-sovellukseen on käytetty JavaScriptiä kaavioiden piirtämiseen.

Työn lopputulos oli vaatimukset täyttävä, mutta kehittämismahdollisuuksiakin on vielä. Muun muassa Arduino mikro-ohjaimen käyttö Raspberry Pi:n lisäksi loisi mahdollisuuksia kehittää järjestelmää.

REMOTE HOME ELECTRICITY CONSUMPTION MONITORING SYSTEM

Juhala, Jeremikael
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Information Technology
December 2017
Supervisor: Ylikoski, Mauri
Number of pages: 25
Appendices: 2

Keywords: remote monitoring, Raspberry Pi, Ruby

The purpose of this thesis was to build a system that reads consumption data from home's electricity energy meter, and from where a user could view this data easily. Requirements for the system were possibility to view the usage information in real time, with a short timeframe and from anywhere.

A Raspberry Pi was used as the hardware of the system. This decision was made so that only one device would be needed, as everything could be done with it. Electricity energy meter's led was used for reading the consumption data. To read the led a photoresistor was used. Python-programming language was used for a program that reads the consumption data. The data was saved into a database in short time intervals to create the possibilities for real time information viewing and information viewing with a short timeframe.

The possibility to view the consumption information anywhere was done by creating a Web-application, as the user can use it wherever as long as they have an Internet connection. The Web-application was realized by using Ruby on Rails application framework and MVC-architecture. In addition, JavaScript was used in the Web-application for drawing diagrams.

The result of the thesis did meet the requirements, but there is still room for further development. For example, using an Arduino microcontroller with the Raspberry Pi would create possibilities for improvement.

SISÄLLYS

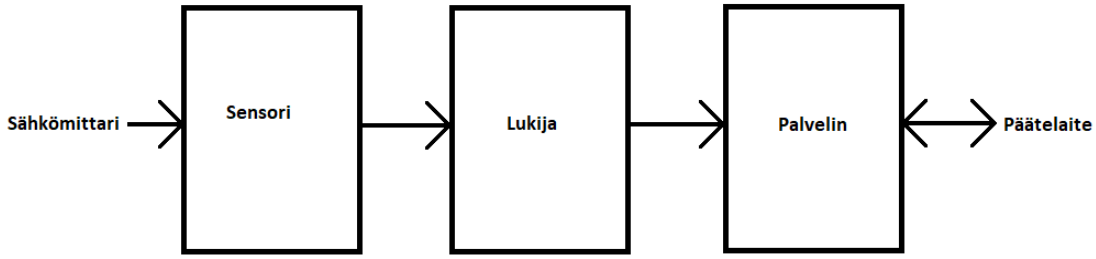
1 JOHDANTO

Monet energiayhtiöt tarjoavat kodin sähkönkulutuksen seuraamisen mahdollistavia palveluita. Nämä palvelut kuitenkin antavat usein vain tunnin tarkkuudella tietoja kulutuksesta. Lisäksi uusin tieto on edelliseltä päivältä. Täten asiakas ei pääse katsomaan, kuinka paljon kulutusta on ollut kuluvana päivänä. Asiakas ei siis myöskään näe paljonko juuri tällä hetkellä on kulutusta.

Tämän opinnäytetyön tarkoituksena on rakentaa kodin sähkönkulutuksen seurantajärjestelmä. Seurantajärjestelmän tulisi antaa reaaliaikaista tietoa mahdollisimman korkealla tarkkuudella. Lisäksi tiedon tulisi olla luettavissa missä ja milloin tahansa älypuhelimen, tabletin tai tietokoneen avulla. Pelkän energiankulutuksen lisäksi tämän järjestelmän pitäisi mahdollistaa myös kulutetun energian hinnan seurannan.

2 SUUNNITTELU

Alla oleva kaavio (kuva 1) esittää työssä vaadittavaa laitteistoa. Sähkömittari ja päätelaite ovat työn ulkopuolelle jääviä laitteita, mutta jotka ovat merkittäviä työn kannalta. Sensorin pitää pystyä lukemaan sähkömittarilta kulutustietoja. Lukija lukee sensorilta tiedon ja tallentaa sen. Palvelimen pitää käsitellä tietoa siten, että se voidaan lähettää päätelaitteelle käyttäjän haluamalla tavalla.



Kuva 1. Lohkokaavio laitteistosta

Kulutustiedon lukeminen sähkömittarilta tapahtuu lediä tarkkailemalla. Mittarin ledi välkkyy kulutuksen mukaan, tietyn monta välähdystä per kilowattitunti. Tätä välkkymistä saa seurattua fotodiodilla, eli valoherkällä diodilla, tai valoherkällä vastuksella. Täten sensori tulee olemaan piiri, jossa on fotodiodi tai valoherkkä vastus.

Toinen tärkeä osa työtä on etäluettavuus mahdollisimman laajalla valikoimalla laitteita. Tämän päätin ratkaista tekemällä oman Web-palvelimen, jolloin käyttäjä pääsisi tietoihin käsiksi mistä vain nettiselaimen avulla. Ainoat vaatimukset tässä tapauksessa lukulaitteelle olisivat nettiselain ja nettiyhteys.

Päätin rakentaa koko järjestelmän Raspberry Pi 3 Model B:tä käyttäen. Tähän lopputulokseen päädyin useammastakin syystä. Suurin syy on se, että Raspberry Pi:llä saan helposti kaikki edellä mainitut asiat suoritettua samalla laitteella. Siis Raspberry Pi toimii sekä lohkokaaviossa esitettynä lukijana, että palvelimena. Raspberry Pi on pieni, yhden piirilevyn tietokone, jossa on lisäksi rivi GPIO -pinnejä (general purpose input and output). Koska kyseessä on tietokone, voi sille asentaa Web-palvelimen. GPIO -pinnien avulla taas voidaan kytkeä sensori Raspberry Pi:hin. Lisäksi hyviä

puolia Raspberry Pi:ssä ovat pieni koko, sisäänrakennettu WLAN antenni ja pieni tehontarve. Raspberry Pi on vain 85.60mm x 56mm x 21mm, eli kutakuinkin saman kokoinen kuin pankkikortti. Täten se mahtuu hyvin sähkömittarin viereen, vaikka tilaa mittarikaapissa ei kovinkaan paljoa olisikaan. WLAN yhteys oli lähes pakollinen, sillä lähimpään ethernet liitäntään olisi ollut todella pitkä matka. Tosin sisäänrakennettu WLAN antenni ei olisi ollut pakollinen, sillä USB kytkentäisiäkin WLAN antenneja löytyy halvalla. Tehoa Raspberry Pi 3 vaatii vain maksimissaankin alle 5 wattia. Pieni tehontarve on taas hyvä, koska laite tulee olemaan päällä aina. (Raspberry Pi Foundation a; Raspberry Pi Foundation b; Raspberry Pi Foundation c; Geerling, J.)

Ohjelmistopuolella päätin käyttää fotodiodin lukemiseen Python ohjelmointikieltä. Tähän päätökseen päädyin kahdesta syystä. Ensimmäinen syy on se, että Pythonille löytyy RPi.GPIO kirjasto. RPi.GPIO kirjaston avulla GPIO -pinnien hallinta on todella helppoa. Toinen syy oli aiempi kokemukseni. Olen käyttänyt Pythonia ennenkin, joten se on jo tuttu minulle.

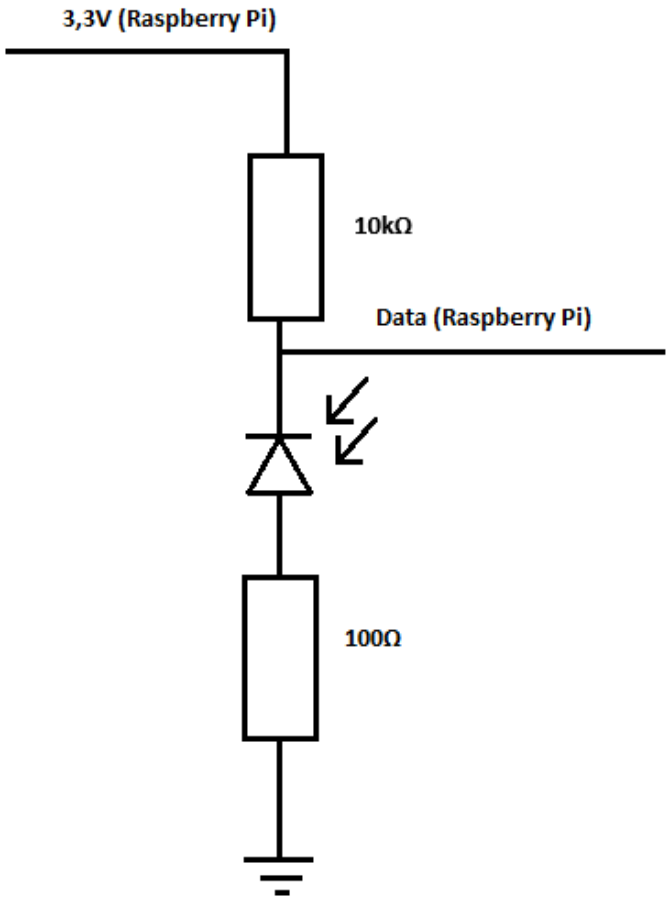
Web-palvelimen ohjelmointiin päätin käyttää Ruby ohjelmointikieltä ja Ruby on Rails ohjelmistokehystä. Tähän päätökseen päädyin lähinnä oman aikaisemman kokemukseni vuoksi. Olen työkseni ohjelmoinut käyttäen Ruby on Railsiä, ja se on mielestäni todella hyvä ohjelmistokehys Web-sovellukselle.

3 KULUTUKSEN LUKEMINEN

3.1 Kytkentä

Kuten aiemmin jo mainitsin, kulutusta saa luettua mittarilta tarkkailemalla mittarissa olevan ledin välkkymistä. Tämä tapahtuu fotodiodin avulla. Fotodiodi toimii pimeässä kuten tavallinenkin diodi, päästäen virran kulkemaan läpi vain yhteen suuntaan. Valon osuessa fotodiodiin, diodi päästää kuitenkin virran kulkemaan molempiin suuntiin. Kun virta kulkee fotodiodin esto suuntaan, toimi diodi siis kuten kytkin, joka on auki pimeässä ja kiinni valon osuessa siihen.

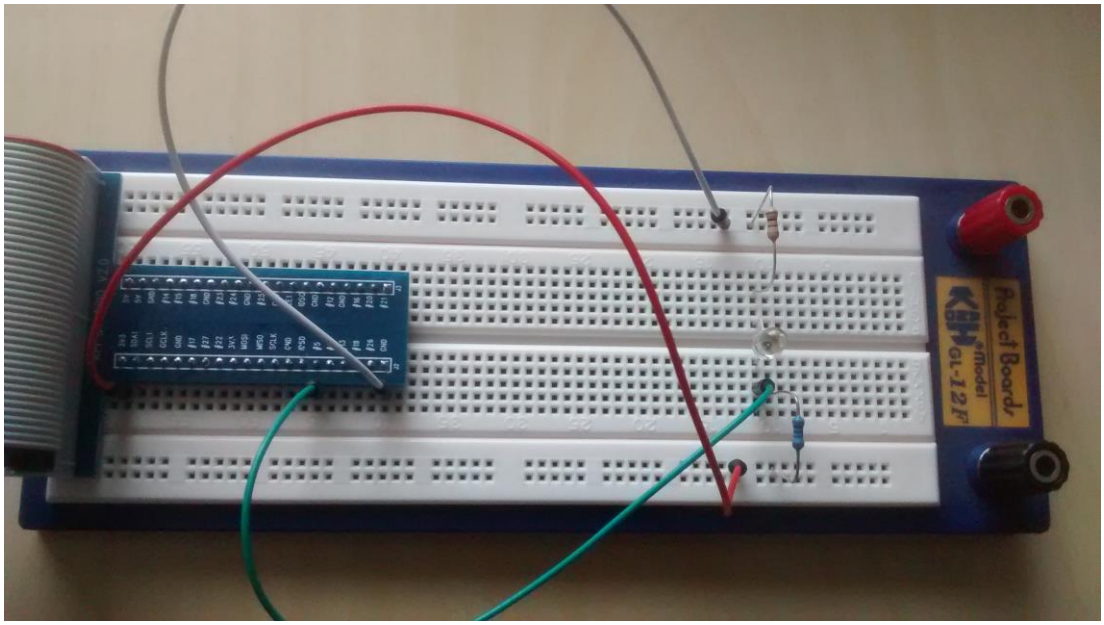
Kytkennässä käytän 3,3 voltin jännitettä, sillä se on Raspberry Pi:n datapinnien ”1”-asento. Päätin kytkeä fotodiodin ylösvetovastusta hyödyntäen. Täten ledin pois päältä ollessa datapinniin tulee 3,3V ja ledin syttyessä datapinni kytkeytyy maahan. Lisäksi kytkin varmuuden vuoksi pienen vastuksen datapinnin ja maan väliin. Tämä pienempi vastus on varmistuksena mahdollisen ohjelmointivirheen varalta. Datapinni voi toimia sekä sisääntulona että ulostulona, mikä määritellään ohjelmallisesti. Jos vahingossa asettaa datapinnin toimimaan ulostulona eikä vastusta olisi maata kohti, tulisi oikosulkutilanne. Piirsin kytkennästä kytkentäkaavion (kuva 2) ennen varsinaista kytkennän rakentamista.



Kuva 2. Fotodiodin kytkentäkaavio

Kytkennän testaamiseksi rakensin ensiksi kytkennän koekytkentälevylle (kuva 3). Ensimmäiseksi testasin kytkentää yleismittarin ja taskulampun avulla. Täten näin miten vastukset eri pisteiden välillä muuttuivat, kun fotodiodille tuli valoa. Tämän

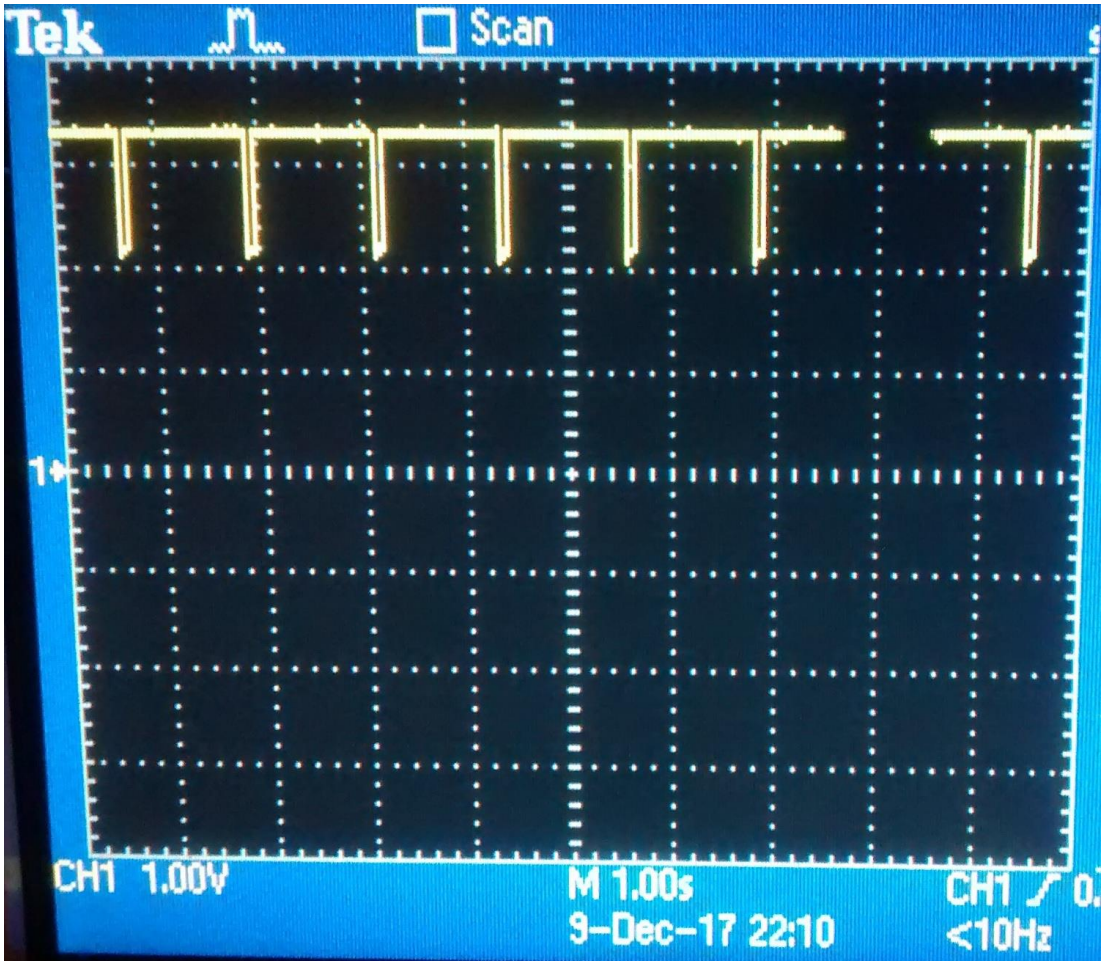
mittauksen perusteella kytkentä toimi halutulla tavalla. Käyttöjännitteen ja maan kytkettyäni sain vielä mitattua jännitetason datapinniltä. Taskulamppua apuna käyttäen sain tulokseksi arvot 3,30V pimeydessä ja 0,22V valon osuessa fotodiodiin. Ideaalitilanteessa nämä arvot olisivat 3,3V pimeydessä ja 0V valossa. Pimeydessä siis päästiin haluttuun tulokseen, mutta valon kanssa jännite jäi hieman yli ideaalitilanteen jännitteen. Tämän pienen eron ei pitäisi kuitenkaan haitata, koska jännitteen sisääntulossa on jonkinlaiset raja-arvot. Virallista tietoa en näistä raja-arvoista löytänyt, mutta Mosaic Industriesin dokumentaatioiden mukaan nämä raja-arvot olisivat 2V ja 0,8V. Täten mikä tahansa jännite alle 0,8 voltin toimisi loogisena ”0”-asentona. (Mosaic Industries)



Kuva 3. Fotodiodi kytkettynä koekytkentälevyllä

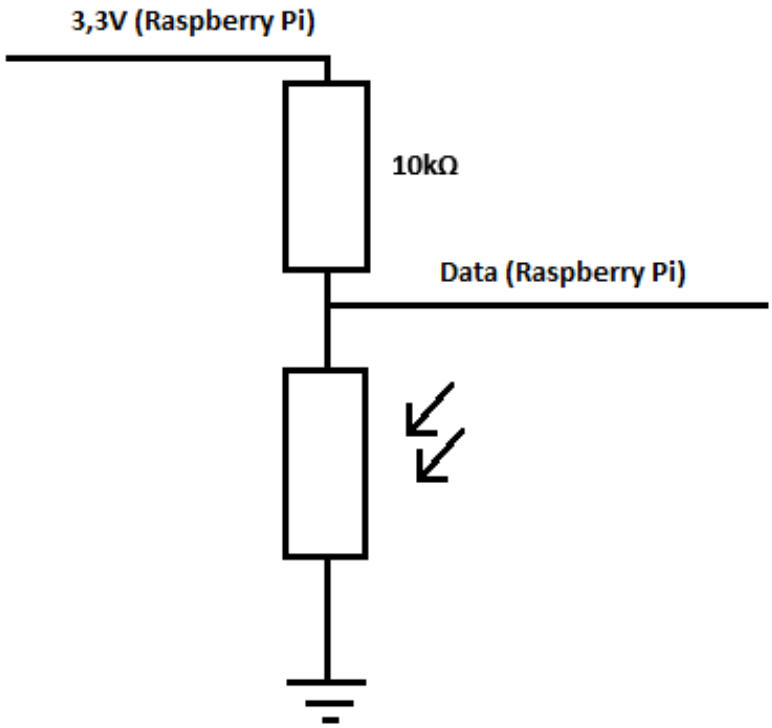
Koska nämä mittaustulokset osoittivat kytkennän toimivan, siirryin testaamaan kytkentää yhdessä Raspberry Pi:n kanssa. Tätä testiä varten tein yksinkertaisen testiohjelman (liite 1). Vielä tässäkin vaiheessa käytin taskulamppua simuloimaan lediä. Tulos oli todella lupaava, sillä kytkentä ja ohjelma molemmat toimivat halutulla tavalla. Seuraavana testissä laitoin kytkennän todelliseen paikkaansa, jotta saisin taskulampun vaihdettua mittarin lediin. Tässä testissä havaitsin ongelman. Ohjelma ei tunnistanut ledin vilkkumista.

Aluksi totesin ongelman syyn olevan mitä todennäköisimmin ledin kirkkaus. Totesin, ettei käytössäni oleva fotodiodi tunnistanut lediä ollenkaan, sillä ledi ei ole tarpeeksi kirkas. Tämä olisi todennäköisesti totta, jos käytössäni oleva fotodiodi toimisi ideaalisen fotodiodin tavoin. Tällöin fotodiodi toimisi valoaktivoituna kytkimenä, mutta todellisuudessa muutos ei ole näin selkeä. Himmeällä valolla fotodiodi päästää virran osittain läpi. Tämän eron huomasin hyvin, kun mittasin kytkentäni toimintaa oskilloskoopilla. Oskilloskoopin piirtämästä kuvasta (kuva 4) näkee hyvin, että jännitetaso datapinnillä muuttuu ledin vilkkuessa, mutta muutos ei ole riittävän suuri. Tämän ongelman olisi voinut korjata esimerkiksi kasvattamalla käyttöjännitteen ja fotodiodin välillä olevan vastuksen suuruutta. Tämä ei kuitenkaan olisi korjannut sitä, että syötän analogista signaalia digitaaliseen sisääntuloon. Tapa, jolla tämäkin korjaantuisi, olisi käyttää komparaattoria, josta kerron tarkemmin luvussa 5 Kehittämismahdollisuudet.



Kuva 4. Oskilloskoopilta saatu kuvaaja fotodiodin toiminnasta sähkömittarin kanssa.

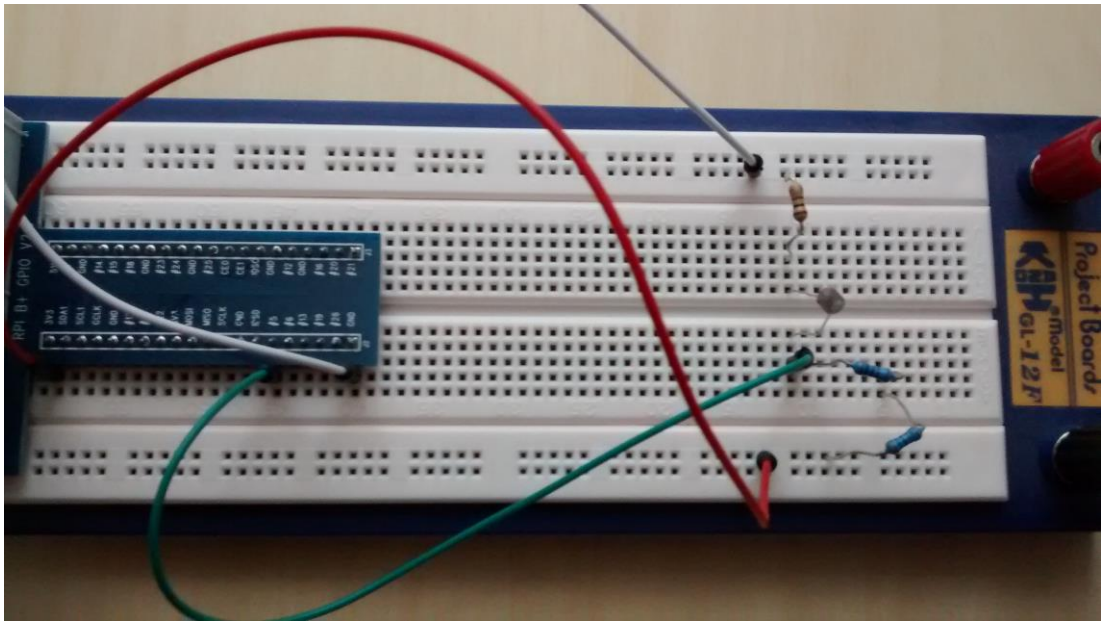
Koska suoritin oskilloskoopilla tehdyn mittauksen vasta myöhemmin, jatkoin työtä oletuksella, että fotodiodin kanssa ei jännitetaso muutu ollenkaan. Siksi päätin kokeilla toimisiko kytkentä mahdollisesti valovastuksen kanssa. Valovastus on kuten normaali vastus, mutta valo vaikuttaa sen vastusarvoon. Kun valovastus on pimeässä, on sen vastusarvo suuri. Kun taas valoa tulee vastukseen, laskee vastusarvo. Valovastuksen käyttö tässä kytkennässä on hyvinkin samankaltaista kuin fotodiodin, joten kytkentää ei tarvitse muuttaa kovinkaan paljoa. Riittää kun vaihtaa fotodiodin paikalle valovastuksen, kuten näkyy kytkentäkaaviosta (kuva 5).



Kuva 5. Kytkentäkaavio valovastusta hyödyntäen

Rakensin jälleen kytkennän ensiksi koekytkentälevylle testaamista varten. Suoritin samat testit kuin fotodiodin kanssa. Tulokset osoittivat, että valovastuksen kanssa kytkentä toimii. Jännitteen kytkettyäni sain yleismittarilla mitattuna tulokseksi 3,30V pimeydessä ja 0,06V valon osuessa valovastukseen. Siis valon osuessa vastukseen jännite laski jopa hieman alemmaksi kuin fotodiodin kanssa. Ero ei kuitenkaan ole merkittävä, etenkin koska mittaukset on tehty taskulamppua kädessä pitäen. Täten valo ei välttämättä osu sensoriin samalla tavalla molemmissa tapauksissa. Jos valo tulee edes hieman viistossa sensorille, muuttuu tulos hieman.

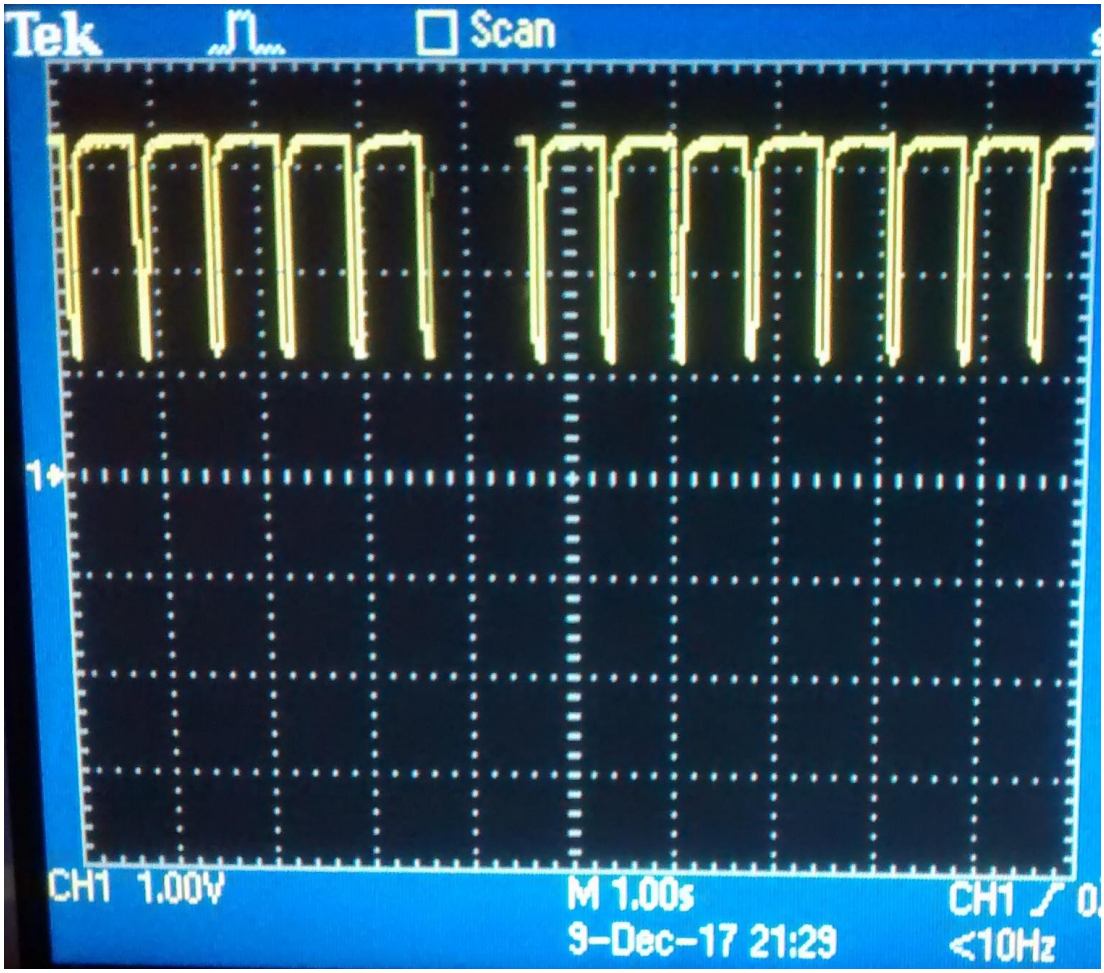
Kytkentä toimi hyvin myös testatessa Raspberry Pi:n kanssa, edellistä testiohjelmaa käyttäen. Ohjelma tunnisti jopa mittarin ledin välkkymisen, vaikkakin oli todella tarkka valovastuksen sijainnista lediin nähden. Koska käytössä on nyt valovastus, ei toiminta ole enää kuten kytkimellä. Sen sijaan valovastuksen arvo muuttuu vähitellen valon kirkkauden muuttuessa. Tästä ominaisuudesta on hyötyä edellä mainitun ongelman korjaamiseksi. Muiden vastusten arvojen muuttamisella saa säädettyä kytkennän herkkyyttä. Kasvattamalla käyttöjännitteen ja datapinnin välissä olevan vastuksen suuruutta saan datapinnin jännitteen lähemmäksi 0 volttia, vaikka valovastuksella olisinkin suuri vastusarvo. Muokkasin koekytkentälevyllä olevaa kytkentää (kuva 6) siten, että kyseisen vastuksen arvo kaksinkertaistui. Tämän jälkeen kokeilin taas kytkennän toimintaa mittarin ledin lukemisessa, ja totesin kytkennän toimivan nyt halutulla tavalla.



Kuva 6. Lopullinen kytkentä koekytkentälevyllä

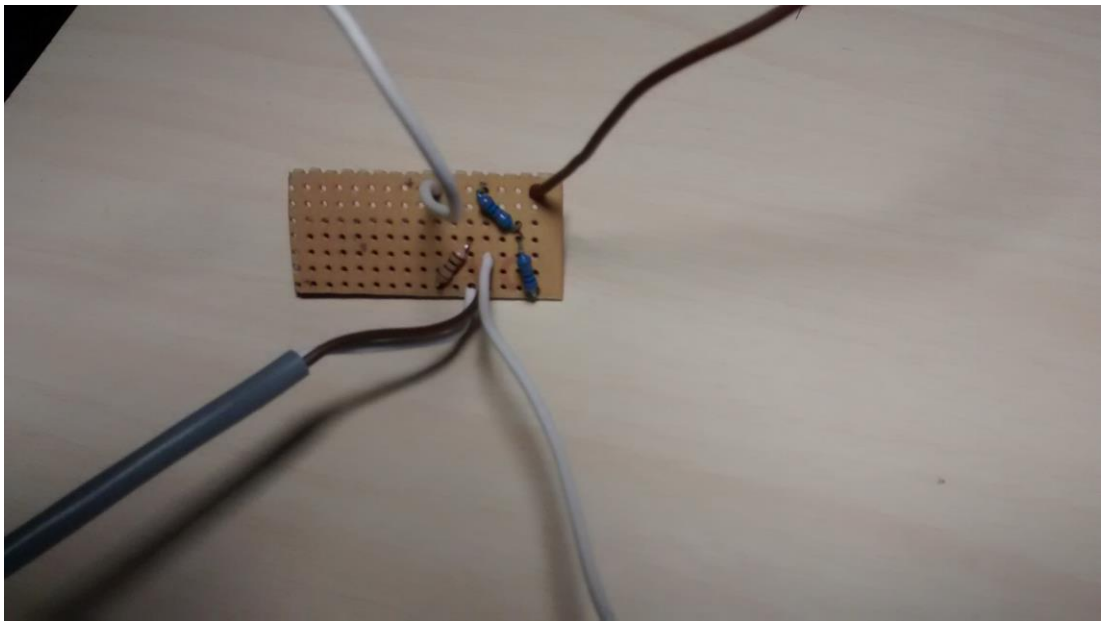
Suoritin myös tämän kytkennän kanssa mittaukset oskilloskoopilla. Tuloksesta (kuva 7) näkee selvästi eroavaisuuden fotodiodin kanssa tehtyyn mittaukseen. Ledin vilkahtaessa piikki menee huomattavasti lähemmäksi nollaa, vaikka jää tässäkin tapauksessa kohtalaisen korkealle, jopa hieman yli 1 voltin. Toinen selvä ero tässä kuvaajassa fotodiodin kuvaajaan verrattuna on pyöreys jännitetason palatessa takaisin 3,3 volttiin. Tämä johtuu valovastuksen hitaudesta. Valovastukset ovat hitaampia

reagoimaan valoisuuden muutokseen kuin fotodiodit, minkä vuoksi valovastus ei tulisi toimimaan tällaisessa käytössä, jos ledi vilkkuisi hyvin nopeasti.

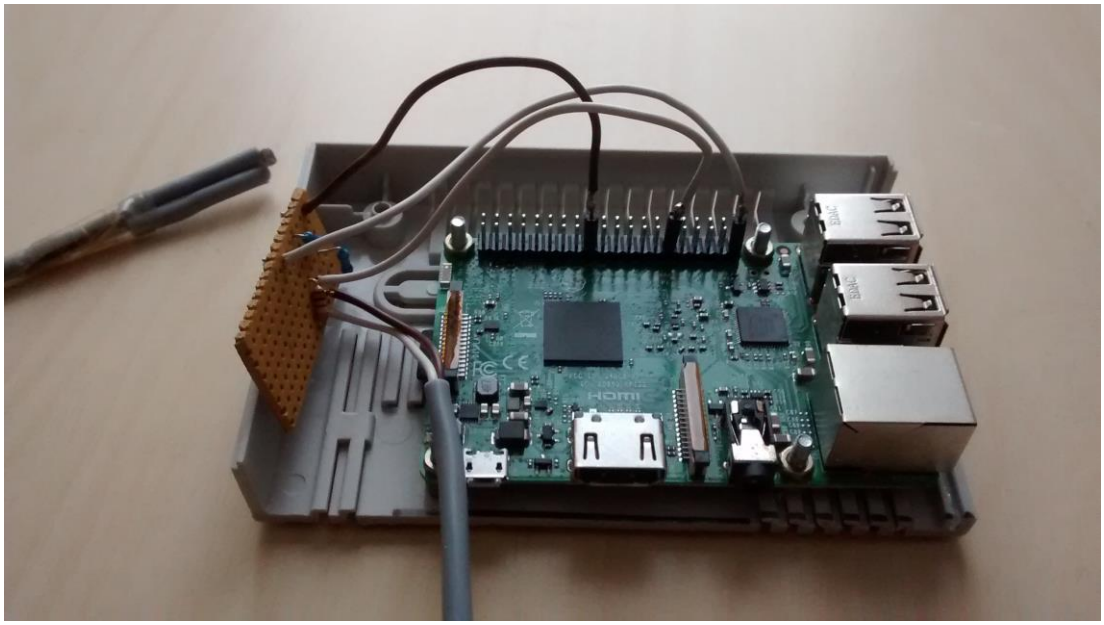


Kuva 7. Oskilloskoopilta saatu kuvaaja valovastuksen toiminnasta sähkömittarin kanssa.

Lopuksi juotin vielä edellä mainitun kytkennän piirilevylle (kuva 8). Testasin juotokset ensin yleismittarilla resistanssiarvoja mitaten. Kun mittaustulokset osoittivat juotokset olevan kunnossa, liitin piirin Raspberry Pi:hin (kuva 9). Testasin kytkennän vielä aiemminkin käyttämälläni testiohjelmalla, ennen laitteen sähkömittariin liittämistä.



Kuva 8. Kytkentä juotettuna piirilevylle



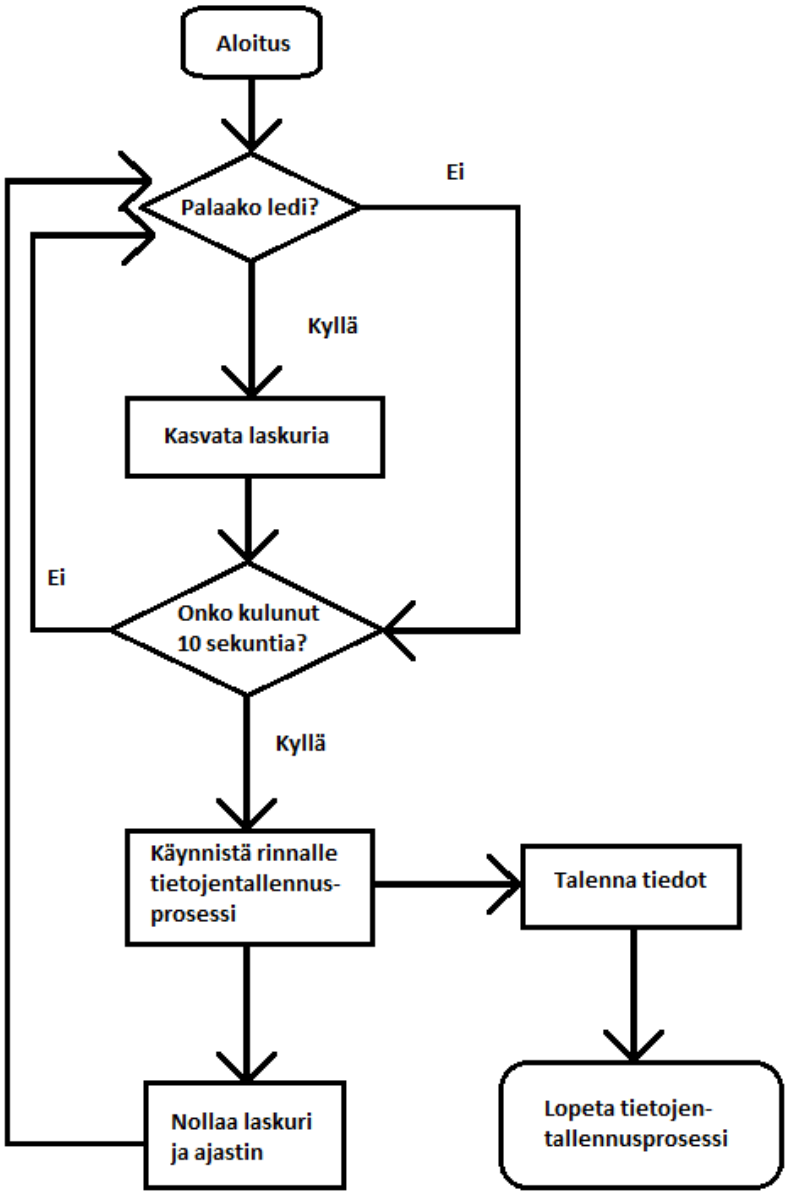
Kuva 9. Lopullinen kytkentä liitettynä Raspberry Pi:hin

3.2 Ohjelma

Lukijaohjelma (liite 2) suorittaa varsinaisen ledin lukemisen yksinkertaisesti jatkuvalla kyselyllä. Loputon silmukka tarkistaa jatkuvasti, onko ledi päällä vai pois päältä, ja kun tässä todetaan, että ledi on päällä, kasvatetaan väläyslaskuria. Tämä olisi parempi suorittaa keskeytyspohjaisena, mutta Raspberry Pi ei tue keskeytyksiä, joten

tämä muutos vaatisi muutoksia myös laitteistopuoleen. Tämä vuoksi päätin tehdä ohjelman loputonta silmukkaa käyttäen.

Joka kymmenes sekunti ohjelma antaa laskurin lukeman, sekä aikavälin, milloin laskuria on laskettu, toiselle prosessille ja nollaa laskurin. Tämä toinen prosessi suorittaa tietojen tietokantaan kirjoittamisen. Tällä moniajolla varmistetaan se, että vaikka tiedon tallentamiseen menisi jostain syystä pitkä aika, saadaan silti kaikki väläykset laskettua. Ohjelman toiminta on kuvattu vuokaaviossa (kuva 10).



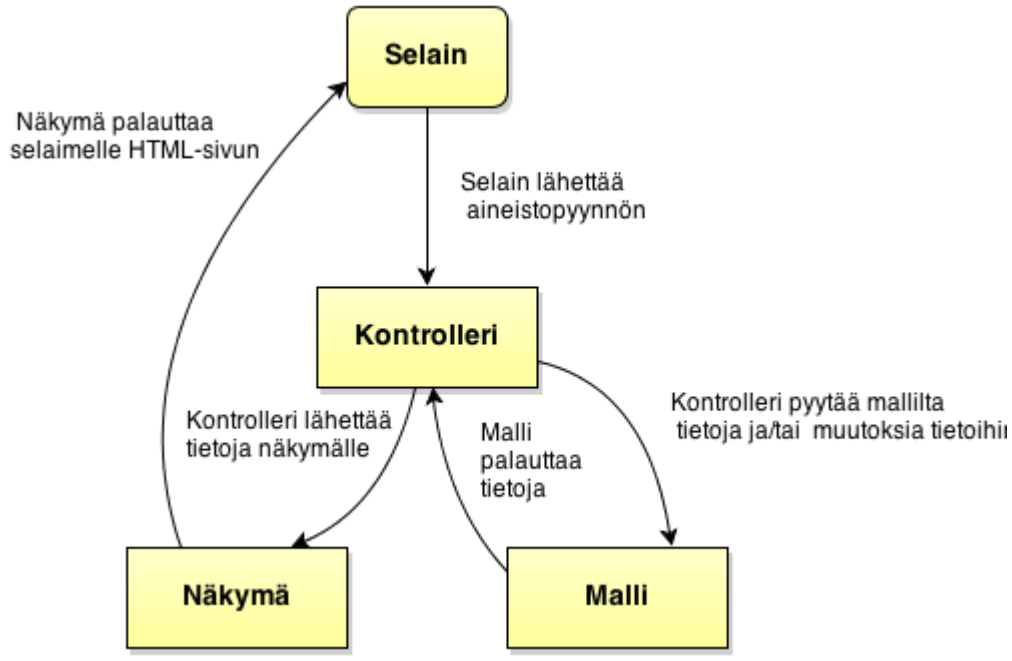
Kuva 10. Lukijaohjelman toiminta.

Lukijaohjelma hyödyntää lisäksi JSON muotoisen asetustiedoston lukemista, jotta tietokantayhteyden muodostamiseen vaadittavat tiedot, sekä montako väläystä on yksi kilowattitunti, eivät ole suoraan ohjelmakoodiin kovakoodattuina. Täten vaikka tulevaisuudessa nämä tiedot vaihtuisivat, ei tarvitse suoraan ohjelmakoodiin tehdä muutoksia.

4 WEB-SOVELLUS

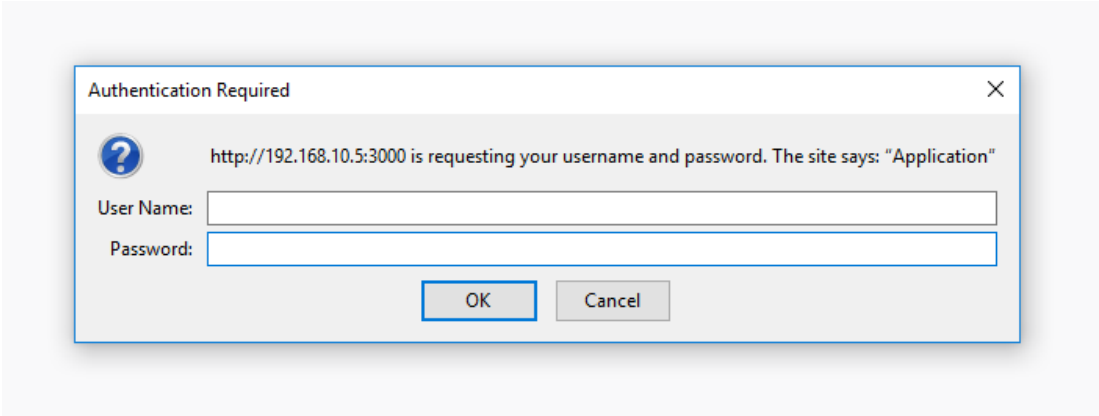
4.1 Yleistä

Web-sovellus on toteutettu Ruby on Rails -ohjelmakehystä ja JavaScriptiä käyttäen. Kuten Ruby on Rails -sovellukset yleensäkin, on tämäkin sovellus toteutettu MVC -arkkitehtuuria noudattaen. MVC on lyhenne sanoista model-view-controller, eli malli-näkymä-käsittelijä. Malli hallitsee sovelluksen tiedon tallentamisen ja hakemisen tietokannasta. Näkymä määrittelee mitä ja miten käyttäjälle näytetään. Käsittelijä, eli kontrolleri, vastaanottaa käyttäjältä kutsut ja hallinnoi mallia ja näkymää niiden perusteella. (Consuegra, D., Laine, H., Saada, S., Lehtola, N. & Suomalainen, L.)



Kuva 11. MVC-malli (Consuegra, D. ym.)

Sovelluksen suojauksen olen suorittanut käyttämällä HTTP Basic -todennusta (kuva 12). Tämä on kaikkein yksinkertaisin todennusmenetelmä. HTTP Basic pyytää käyttäjältä käyttäjätunnusta ja salasanaa, mutta nämä ovat sivuston yleiset käyttäjätunnus ja salasana. Käyttäjätilejä ei siis käytetä ollenkaan. Tähän todennusmenetelmään päädyin, koska kyseisen sovelluksen käyttäjäkunta on todella pieni, joten käyttäjätilien hallinnasta ei olisi saatu todellista hyötyä.



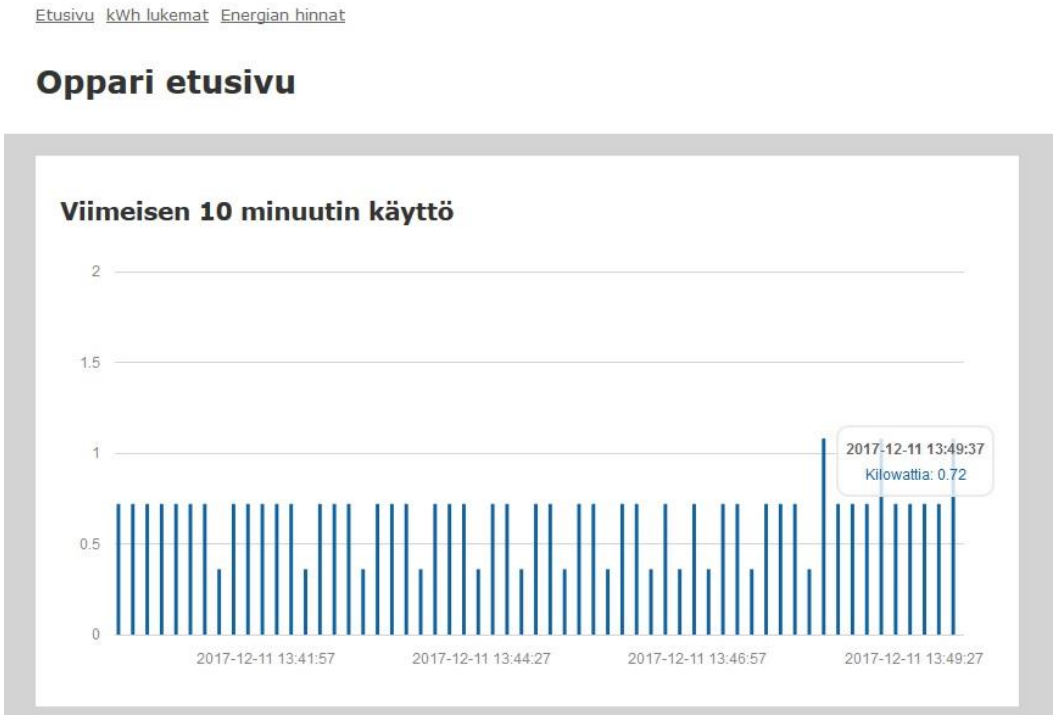
Kuva 12. HTTP Basic -todennus

4.2 Etusivu

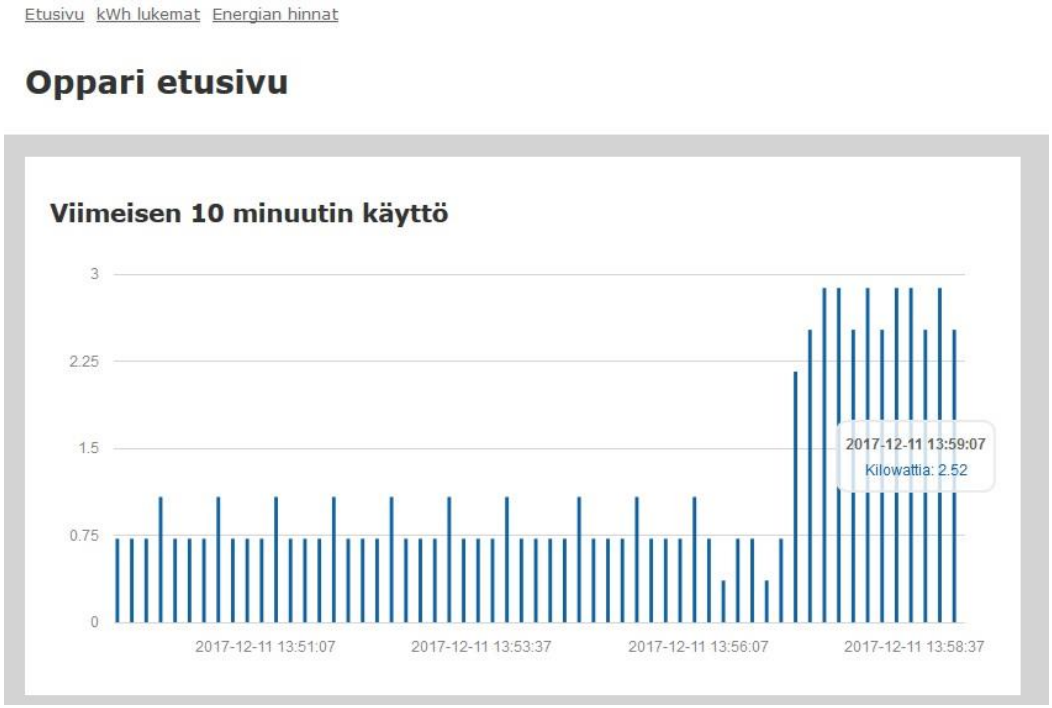
Etusivulle on koottu kaikki käyttäjää normaalissa päivittäisessä käytössä kiinnostavat tiedot. Siitä löytyy kolme pylväsdiagrammia eri tietoja esittämään. Toteutin diagrammit JavaScriptillä, jotta ne olisivat responsiivisia, sillä palvelimen päässä piirretty diagrammi pitäisi näyttää käyttäjälle vain normaalina kuvana. Päädyin toteuttamaan diagrammit morris.js -kirjastolla, sillä se vaikutti helppokäyttöiseltä, ja ominaisuuksiltaan riittävältä.

Jotta käyttäjä näkisi heti sivulle tullessaan kyseisellä hetkellä olevan käytön, on etusivulla ensimmäisenä diagrammi esittämässä tämänhetkistä käyttöastetta (kuva 13). Siinä näytetään kymmenen sekunnin tarkkuudella keskimääräinen teho kilowatteina viimeisen kymmenen minuutin ajalta. Kuvassa 14 näkyy hyvin, kuinka kulutus kasvaa, kun uuni laitetaan päälle. Molemmissa kuvissa on havaittavissa suhteellisen isoa heittelyä yksittäisten mittausten välillä. Tämä johtuu siitä, että käytössä olevan sähkömittarin ledin välähtely näyttää vain yhden wattitunnin tarkkuudella kulutuksen. Täten ei kymmenessä sekunnissa ehdi tulemaan montaa välähdystä, jolloin yhdenkin

välähdyksen ero näyttää kuvaajassa suurelta. Tämä olisi varmasti hyvä korjata tulevaisuudessa, esimerkiksi muuttamalla sovellusta näyttämään tiedot vain pidemmän aikavälin tarkkuudella.



Kuva 13. Viimeisen kymmenen minuutin käyttö -kaavio normaalilla käytöllä



Kuva 14. Uunin päälle laitto näkyy kaaviossa

Jälkimmäiset kaksi diagrammia (kuva 15) esittää historiatietoja. Ensimmäinen näistä kahdesta, eli vihreällä esitetty diagrammi, esittää energian kulutusta kilowattitunteina. Jälkimmäinen, eli punaisella esitetty diagrammi esittää energian hintaa.

Kuva 15. Historia-kaaviot

Vakiona historiadiagrammit näyttävät historian tunnin tarkkuudella viimeisen vuorokauden ajalta, mutta käyttäjä saa itse päätettyä aikavälin ja tarkkuuden diagrammien yllä olevasta valinnasta. Päätettävänä ovat historian aloitus- ja lopetuspäivämäärä, sekä tarkkuus, jolla historia näytetään. Vaihtoehtoina tarkkuuteen ovat tällä hetkellä tunti, päivä ja kuukausi, mutta tulevaisuudessa sovellusta kehittäessä voi niitä helposti lisätä.

4.3 kWh lukemat -sivu

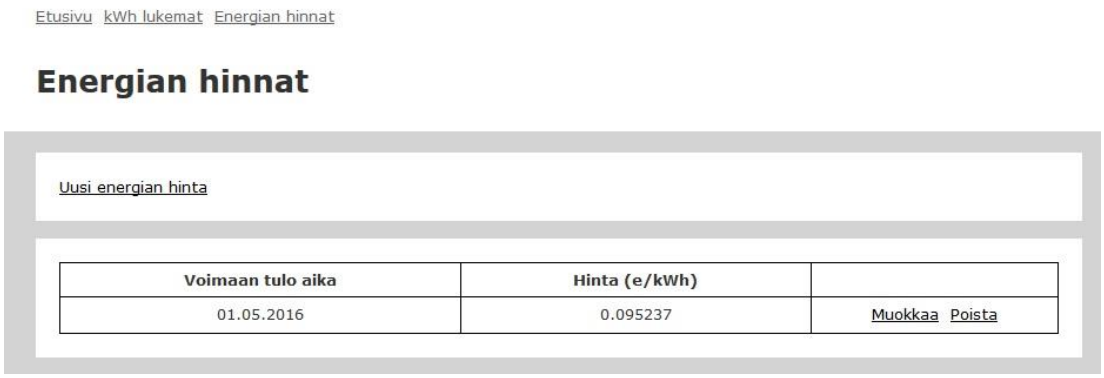
Toisena sivuna on kWh lukemat -sivu (kuva 16). Tällä sivulla näkyy taulukko kaikista lukijaohjelman tallentamista merkinnöistä. Merkinnöistä näytetään mittauksen aloitus- ja lopetusajankohta, energian kulutus wattitunteina, sekä keskimääräinen teho kilowatteina.

Kuva 16. kWh lukemat -sivu

Koska mittausväli on kymmenen sekuntia, on merkintöjä näytettäväksi todella paljon. Jotta sivu ei kasvaisi liikaa, on tiedot sivutettu käyttäen Kaminari-gemiä. Kaminarin käyttöön päädyin sen helppokäyttöisyyden, sekä aiemman kokemuksen perusteella. Yhdellä sivulla näytetään vain viisikymmentä merkintää, ja sivun alareunassa on sivun valinta linkit.

4.4 Energian hinnat -sivu

Energian hinnat -sivulta (kuva 17) löytyvät energian hintatasot eri aikoina. Hinnat pitää käyttäjän käydä itse lisäämässä, asettaen sille voimaantuloajan ja hinnan, muodossa euroa per kilowattitunti. Näitä hintoja käytetään etusivun hinta historia diagrammin hintojen laskemiseen.



Kuva 17. Energian hinnat -sivu

Sivulla on ensimmäisenä linkki uuden hinnan luomiskaavakkeeseen. Tämä on ensimmäisenä, jotta se ei menisi piiloon tulevaisuudessakaan, kun hintoja on kertynyt enemmän. Alta löytyy taulukko, jossa näkyy aikaisemmin asetetut energian hinnat, ja linkit merkinnän muokkaamiseen ja poistamiseen.

Energian hinnan luominen tapahtuu pienellä, yksinkertaisella kaavakkeella (kuva 18). Kaavakkeesta käyttäjä valitsee pudotusvalikoita käyttäen hinnalle voimaantulopäivämäärän ja kirjoittaa hinnan euroina per kilowattitunti. Myös hintatiedon muokkaussivu käyttää samaa kaavaketta, ja siten näyttää otsikkoa lukuun ottamatta samalta.

Kuva 18. Energian hinnan lisäys sivu

Käyttäjän ei ole kuitenkaan tarkoitus muokata taikka poistaa hintatietoja, paitsi mahdollisen virheen tehtyään. Täten hintahistoriassa saadaan näytettyä todelliset arvot aina oikeilla, kyseisenä ajankohtana voimassa olleilla hinnoilla.

5 KEHITTÄMISMAHDOLLISUUDET

Työtä tehdessä mieleen tuli muutamia parannusmahdollisuuksia, joita en kuitenkaan toteuttanut. Näistä suurimmat liittyy ledin lukemiseen. Koska en saanut kytkentää toimimaan fotodiodilla, käytin sen tilalla valovastusta. Valovastus ei kuitenkaan ole yhtä nopea reagoimaan kirkkauden muutokseen, kuin fotodiodi. Käytössäni olevalla kilowattituntimittarilla tämä ei haittaa, sillä sen ledi vilkkuu suhteellisen hitaasti. Monessa kilowattituntimittarissa kuitenkin ledi vilkkuu nopeammin. Jos tämän työn haluaisi toimivan millä tahansa mittarilla, pitäisi siis käyttää fotodiodia.

Fotodiodin saisi toimimaan ledin kanssa, jos hyödyntäisi komparaattoria kytkennässä. Komparaattori on piiri, joka vertailee kahta jännitettä toisiinsa, ja palauttaa digitaalisen signaalin. Tämä signaali kertoo kumpi, kahdesta jännitteestä on suurempi. Komparaattorilla saisi siis pienemmänkin jännite-eron tunnistettua. Lisäksi komparaattoria käyttämällä saisi todellisen digitaalisen signaalin, toisin kuin nykyisestä käyttämästäni piiristä. Nykyisellään piiriltä tulee analoginen signaali, jonka Raspberry Pi tulkitsee digitaaliseksi.

Myös lukijaohjelmaa voisi kehittää. Jos ohjelman muuttaisi toimimaan keskeytyspohjaiseksi, ei tarvitsisi kokoaikaa kysellä onko ledi päällä, vaan kun ledi syttyy, tulisi ohjelmalle keskeytys. Täten ohjelman ajaminen kuluttaisi vähemmän resursseja Raspberry Pi:ltä. Keskeytyspohjainen sovellus toimisi myös varmemmin siten, että jokainen ledin välähdys tulisi varmasti havaittua. Nyt saattaisi käydä siten, että ohjelma on väärässä kohdassa suoritustaan ledin välähtäessä, jolloin kyseinen väläys jäisi havaitsematta.

Esimerkiksi Arduinoissa olisi sisäänrakennettuna komparaattori, sekä mahdollisuus käyttää keskeytyksiä. Työhön voisi siis liittää Arduinon sensoripiirin ja Raspberry Pi:n väliin, ja ohjelmoida Arduinolle lukijaohjelma. Näin saisi kaikki edellä mainitut parannukset tehtyä työhön.

Myös Web-sovelluksen puolella on kehitettävää. Käyttökokemusta voisi sulavoittaa hyödyntämällä enemmän JavaScriptiä. Esimerkiksi etusivun nykyistä käyttöä esittävä diagrammi voisi päivittyä automaattisesti, jos JavaScriptillä haettaisiin tietyin aikavälein uudet tiedot. Myös etusivun historian aikaväliä tai tarkkuutta muuttaessa voisi suorittaa diagrammien päivityksen JavaScriptiä hyödyntämällä. Täten ei tarvitsisi aina ladata koko sivua uudelleen.

Vaikka Web-sovellus on tehty toimimaan mobiililaitteilla, voisi järjestelmää kehittää esimerkiksi tekemällä kokonaan oman mobiilisovelluksen. Täten mobiililaitteille saataisiin luotua varmasti paras mahdollinen käyttökokemus.

Web-sovelluksessa ei myöskään ole kielivaihtoehtoja, vaan kaikki tekstit ovat vain suoraan suomeksi kirjoitettuja. Kuitenkin jos työtä haluaisi tuotteistaa, pitäisi varmasti käännökset tehdä. Tämän voisi tehdä helposti I18n-gemillä, joka löytyy Ruby on Rails sovelluksista vakiona, mutta jota ei ole tässä työssä hyödynnetty.

Lisäksi pitäisi toteuttaa käyttäjien hallinta, eikä rajata pääsyä pelkästään HTTP Basic -todennuksen avulla.

6 YHTEENVETO

Työn tavoitteena oli rakentaa kodin sähkönkulutuksen etäseurantajärjestelmä. Järjestelmän piti lukea kulutustietoja kodin sähkömittarilta. Näihin lukemiin oli tarkoitus päästä käsiksi mistä tahansa, millä laitteella tahansa. Lukemia oli tarkoitus voida lukea korkealla tarkkuudella. Lisäksi piti saada tiedot sähkönkulutuksen hinnasta. Mielestäni onnistuin tavoitteissa hyvin.

Työn lopputulos lukee sähkömittarilta tietoja, joita sitten pääsee lukemaan etänä. Vaatimuksia luentalaitteelle on vain kaksi: nettiyhteys ja nettiselain. Tosin, vaikka sovellusta voi käyttää älypuhelimella, ei käyttöliittymä ei ole kovin hyvä pienen näytön vuoksi.

Lukutarkkuus on todella korkea. Käyttäjä pääsee seuraamaan nykyistä kulutusta kymmenen sekunnin tarkkuudella. Myös kulutushistoriaa pääsee katsomaan tunnin, päivän tai kuukauden tarkkuudella. Työn alussa oli tarkoitus olla myös historia saatavilla tarkemmin, mutta työtä tehdessä totesin, ettei todellisuudessa tarvetta ole. Tietenkin tulevaisuudessa voi lisätä tarkkuusvaihtoehtoja historiaan, jos tarve tulee.

Myös hintahistoria löytyy Web-sovelluksesta, kuten alkuperäisiin tavoitteisiin kuuluikin.

Tietenkin aina löytyy mahdollisuuksia parantaa ja jatkokehittää järjestelmää, kuten kappaleesta 5 Kehittämismahdollisuudet käy ilmi. Kuitenkin työn lopputulos on alkuperäisten tavoitteiden mukainen, joten olen tyytyväinen siihen.

LÄHTEET

Consuegra, D., Laine, H., Saada, S., Lehtola, N. & Suomalainen, L. Arkkitehtuuri ja MVC. Viitattu: 11.12.2017. Saatavissa: https://advancedkittenry.github.io/koodaaminen/arkkitehtuuri/index.html

Geerling, J. Power Consumption. Viitattu 30.4.2017. Saatavissa: https://www.pidramble.com/wiki/benchmarks/power-consumption

Mosaic Industries. GPIO Electrical Specifications Viitattu: 11.12.2017. Saatavissa: http://www.mosaic-industries.com/embedded-systems/microcontroller-projects/raspberry-pi/gpio-pin-electrical-specifications

Raspberry Pi Foundation a. Raspberry Pi 3 Model B. Viitattu 30.4.2017. Saatavissa: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

Raspberry Pi Foundation b. GPIO: Models A+, B+, Raspberry Pi 2 B and Raspberry Pi 3 B. Viitattu 30.4.2017. Saatavissa: https://www.raspberrypi.org/documentation/usage/gpio-plus-and-raspi2/

Raspberry Pi Foundation c. FAQS. Viitattu 30.4.2017. Saatavissa: https://www.raspberrypi.org/help/faqs/

```
#!/usr/bin/env python
from time import sleep
import RPi.GPIO as GPIO, time

# Setup GPIO pins
GPIO.setmode(GPIO.BCM)
data_pin = 5
GPIO.setup(data_pin, GPIO.IN)

counter = 0

# Run loop until user gives a keyboard interrupt
try:
  while True:
    # If input from GPIO pin is false, it means the LED is on
    if GPIO.input(data_pin) == False:
      # Print something, to tell the user the LED flash was detected
      # Counter helps to see a new flash was detected, if the program is ran for a while
      counter += 1
      print '{} {}'.format('flash!', counter)

      # Wait a while to not get multiple prints on one flash
      time.sleep(0.1)



# Software is closed with keyboard interrupt,
# after which GPIO setups are reseted
except KeyboardInterrupt:
  GPIO.cleanup()
```

```
#!/usr/bin/env python
from time import sleep
from datetime import datetime
import RPi.GPIO as GPIO, time
import MySQLdb
from multiprocessing import Pool
import json

def save_to_db(start, stop, flashes):
  # Read configs from separate file
  with open('config.json') as config_file:
    config = json.load(config_file)

  flashes_per_kwh = config['flashes_per_kwh']
  start = start.isoformat()
  stop = stop.isoformat()

  # Create a database connection
  db = MySQLdb.connect( host=config['database']['host'],
                        user=config['database']['user'],
                        passwd=config['database']['passwd'],
                        db=config['database']['db'] )
  cur = db.cursor()

  # Calculate the actual kwh reading, instead of number of flashes
  kwh = flashes / float(flashes_per_kwh)

  # Save info to database
  cur.execute("""INSERT INTO kwh_readings (started_at, stopped_at, value, created_at, updated_at)
              VALUES (%s, %s, %s, %s, %s);""", [start, stop, kwh, stop, stop])
  db.commit()
```

```
  # Close the db connection
  db.close()


# Setup GPIO pins
GPIO.setmode(GPIO.BCM)
data_pin = 5
GPIO.setup(data_pin, GPIO.IN)

counter = 0
start = datetime.now()

# Create a separate porcess for saving info in database
db_pool = Pool(processes=1)

# Run loop until user gives a keyboard interrupt
try:
  while True:
    # If input from GPIO pin is false, it means the LED is on
    if GPIO.input(data_pin) == False:
      # Count the number of flashes
      counter += 1

      # Wait a while to not count one flash as multiple
      time.sleep(0.1)


    stop = datetime.now()
    if (stop - start).total_seconds() >= 10:
      # Use the separate process to save info, and don't wait for it,
      # so we don't miss any flashes while saving
      db_pool.apply_async(save_to_db, [start, stop, counter])
```

```
        # Reset values
        start = stop
        stop = None
        counter = 0


# Software is closed with keyboard interrupt,
# after which GPIO setups are reseted
except KeyboardInterrupt:
  GPIO.cleanup()
```