Opinnäytetyö (AMK)

Tieto- ja viestintätekniikka

2020

Arttu Girs

RAJAPINNAN LUOMINEN WITHINGS API:A KÄYTTÄEN

TURKU AMK

TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniikka

2020 | 36 sivua

Arttu Girs

RAJAPINNAN LUOMINEN WITHINGS API:A KÄYTTÄEN

Click here to enter text.

Opinnäytetyön tavoitteena oli kehittää Turun ammattikorkeakoulun uudelle Healt Tech Labille eli terveysteknologian laboratoriolle verkkosivun tyyppinen rajapinta, joka automaattisesti keräisi, päivittäisi ja esittäisi laboratorion Withings-merkkisillä laitteilla saadun datan. Työn toissijainen tavoite oli yleinen tutustuminen Withings laitteisiin ja niiden käyttöönotto.

Laboratorion Withings laitteet ovat BPM Core -verenpainemittari, Sleep -unimatto ja Thermo -älylämpömittari. Withings on kehittänyt oman alustansa, sovelluksen nimeltään Healthmate, johon kerääntyy yhden käyttäjän kaikilla eri laitteilla saatu data. Healt Tech Labissa kuitenkin haluttiin oma rajapinta, joka keräisi kaiken datan yhteen paikkaan ilman, että se menisi Healthmaten kautta.

Työn toteutuksessa käytettiin enimmäkseen kolmea eri teknologiaa: Withings API:a, Node-Redia ja Reactia. API tarkoittaa sovellusohjelmointirajapintaa ja Withings API on luotu tarjoamaan kehittäjille pääsy eri käyttäjien keräämiin tietoihin Withings laitteilla, ja juuri siihen sitä käytettiin myös tässä opinnäytetyössä. Kun käyttäjän dataan on päästy käsiksi Withings API:n avulla, käytettiin Node-Red nimistä ohjelmointityökalua, jolla yhdistettiin Withings API ja luotu rajapinta. Node-Redin avulla siis voitiin lähettää saatu data toiselle ohjelmalle, joka taas viimeistelisi sen muotoilun ja tulostaisi sen rajapinnan näytölle.Lopuksi itse rajapinta ja ohjelmonti tehtiin React-nimisellä ohjelmointikehikolla, jolla yksinkertaisesti otettiin yhteys Node-Redin lähettämään dataan ja tulostettiin se näytölle pienen muotoilun jälkeen.

Kaiken kaikkiaan opinnäytetyön tavoite, eli rajapinnan luominen onnistui hyvin. Se hoitaa perustehtävänsä hyvin, mutta se vaatii tällä hetkellä liikaa ei toivottua ylläpitotyötä. Siinä on myös vielä tilaa jatkokehitykselle ja yleiselle kehitykselle, mutta siitä huolimatta, siitä on varmasti hyötyä toimeksiantajalle.

ASIASANAT:

Rajapinta, Withings, API, Node-Red, React, laitteistokehitys

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Informations and communications technology

2020 | 36 pages

Arttu Girs

CREATING AN INTERFACE WITH THE WITHINGS API

Click here to enter text.

The objective of this thesis was to develop an interface, resembling a website, for the Health Tech Lab of Turku University of Applied Sciences. The purpose of this interface would be to automatically gather, update and present data gathered with the lab's Withings branded devices. The secondary purpose of this thesis was to more generally become familiar with these devices and initially deploy them.

The Withings devices in the laboratory are the BPM Core blood pressure monitor, the Sleep tracking mat and the smart thermometer. Withings has previously already developed their own platform, an app called Healthmate, which gathers all the data a single user has generated with all their Withings devices, but the Health Tech Lab wanted their own interface, which would collect all the data, but without going through Healthmate.

Three different technologies were used in the execution of this project: The Withings API, Node-Red and React. API stands for for application programming interface, and the Withings API was created to give developers access to certain data of different users and that was exactly what it was used for in this thesis. After access to the users data had been acquired with the help of Withings API, a programming tool called Node-Red was used to connect the API with the created interface. So in essence, Node-Red was used to send the acquired data to another program which would in turn properly format it and print it out to the screen of the interface. Finally, the interface itself was programmed using the React programming framework. React was used to simply connect to the data sent by Node-Red and after small adjustments, display it on the screen.

All in all, the objective of the thesis, which was to create the interface, was successful. The interface handles it's basic task well, although it occassionally requires some unwanted maintenance. It also has some room for further development and some refining but regardless it should be of benefit to the client.

KEYWORDS:

Interface, API, Withings, Node-Red, React, software development

SISÄLTÖ

KUVAT

KÄYTETYT LYHENTEET JA SANASTO

API	Lyhenne sanoista Application Programming Interface. Suomeksi ohjelmointirajapinta.
Curl	Komentorivityökalu, jota käytetään tiedonsiirrossa. [3]
Healthmate	Withingsin kehittämä mobiilisovellus, joka kerää käyttäjän kaiken Withings laitteilla tuottaman datan.
HTML	Lyhenne sanoista Hypertext Markup Language. Kuvausikieli, jota käytetään verkkosivujen luomisessa.
http	Lyhenne sanoista Hypertext Transfer Protocol. Protokolla, jota käytetään tiedon suojatussa siirrossa.
JavaScript	Ohjelmointikieli, jota HTML käyttää ja jota käytetään verkkosivujen luomisessa.
JSON	Lyhenne sanoista JavaScript Object Notation. Avoimen standardin tiedonvälitysmuoto.
JSX	Laajennus normaaliin JavaScript kieleen, jota käytetään React ohjelmointikehikossa. Lyhenne sanoista JavaScript XML. [20]
Komponentti	Uudelleenkäytettäviä koodin osia React ympäristössä.
Node.js	Ympäristö, jonka avulla JavaScript koodia voidaan suorittaa palvelimella.
Node-Red	Node.js pohjainen visuaalinen ohjelmointityökalu. [10]
Oauth	Avoin standardi, jonka avulla voi hallinnoida pääsyoikeuksia tiettyihin palveluihin. Mahdollistaa sisään kirjautumisen palveluihin toisen palvelun, kuten Googlen tai Facebookin, tunnusten kanssa. [1]
Ohjelmointikehikko	Kehitysohjelmisto, joka antaa käyttöön mm. ohjelmointikirjastoja ja työkaluja, antaen näin tukea ohjelmointiin. React on esimerkki tällaisesta kehikosta. (engl. framework)
PHP	Hypertext Preprocessor. Ohjelmointikieli, jota käytetään yleisesti verkkosivujen luomiseen. [12]
React	JavaScript kirjasto ja ohjelmointikehikko, jota käytetään yksisivuisten verkkosivujen tai rajapintojen luomisessa. Tunnetaan myös nimillä Reactjs ja React.js. [13]
REST	Lyhenne sanoista Representational State Transfer. API:n kehityksen arkkitehtuurinen tyyli. [23]
RESTful	API, joka on tyyliltään REST:in mukainen.

SOAP	Lyhenne sanoista Simple Object Access Protocl. Virallinen protokolla, joka standardoi mm. API:en tiedonsiirtoa. [24]
Solmu	Koodia sisältävä, tietyn tehtävän omaava palanen, joista Node-Red sovellus koostuu. (engl. node)
URI	Lyhenne sanoista Uniform Resource Identifier. Merkkijono, jonka avulla kerrotaan tietyn tiedon sijainti järjestelmässä. URL (Uniform Resource Locator) on eräänlainen URI, joka osoittaa verkkosivun sijainnin internetissä.
Visual Studio Code	Koodin muokkaus työkalua ja ympäristö.
Vuo	Kun useita Node-Red solmuja yhdistetään toisiinsa, yhdessä ne luovat vuon. (engl. flow)
XML	Lyhenne sanoista Extensible Markup Language. Standardi, jota käytetään formaattina tiedonvälityksessä.

1 JOHDANTO

Opinnäytetyön toimeksiantajana toiminut Health Tech Lab on Turun ammattikorkeakoulun ICT-Cityn tiloissa, osana terveysteknologian tutkimusryhmää toimiva, test bed -ympäristö, joka tarjoaa palveluja terveysteknologian koulutukseen ja tuotekehitykseen. [19] Health Tech Lab on toimintaympäristönä varsin uusi. Se avautui virallisesti vuoden 2019 syyslukukauden alussa, joten Turun ammattikorkeakoulun opiskelijoilla on ollut mahdollisuus auttaa sen toiminnan kehittämisessä esimerkiksi harjoittelujen tai opinnäytetöiden kautta. Laboratorioon on hankittu useita erityyppisiä laitteita, kuten OpenBCI-aivokäyräpäähine, Oura-älysormuksia ja kolme Withings-nimisen yrityksen laitetta, joiden pohjalta tämä opinnäytetyö tehtiin. Withings on ranskalainen elektroniikkayritys, joka on erikoistunut verkkoon liitettävien laitteiden kehitykseen ja innovointiin. [28] Health Tech Labista löytyvät Withings-laitteet ovat älykäs BPM Core -verenpainemittari, Sleep unenseurantamatto ja älykäs Thermo -lämpömittari.

BPM Coressa on verenpainemittarin lisäksi myös EKG ja digitaalinen stetoskooppi, jotka voivat havaita tiettyjä sydänläppäsairauksia ja eteisvärinän. Nämä ovat molemmat tyypillisiä sydän- ja verisuonitauteja korkeasta verenpaineesta kärsiville. [28, 29] Unimatto taas seuraa muun muassa unen eri vaiheiden eli syvän-, kevyen- ja REM-unen pituutta, sydämen sykettä ja kuorsauksen kestoa. Kuumemittari puolestaan mittaa ruumiin- ja ihonlämpöä. [28]

Tämän opinnäytetyön toimeksiantona oli kehittää Turun ammattikorkeakoulun Health Tech Labille eli terveysteknologian laboratoriolle verkkosivuntyylinen rajapinta, joka automaattisesti kerää, esittää ja päivittää tietyillä laboratorion laitteilla kerättyjä tietoja. Työ koostuu kolmesta eri vaiheesta, jotka toimivat yhteydessä toisiinsa ja tuottavat lopullisen kokonaisuuden. Nämä kolme osaa ovat laboratorion laitteilla kerätyn datan hakeminen ja siihen käsiksi pääsy Withings API:n avulla, tämän datan käsittely Node-Red ohjelmointityökalun kanssa ja itse rajapinnan luominen React-ohjelmointikehikon avulla.

Withings on myös kehittänyt oman mobiilisovelluksena, nimeltään Healthmate, johon käyttäjä voi yhdistää kaikki omistamansa Withings-laitteet, minkä jälkeen niillä kerätty data siirtyy suoraan sovellukseen. Toimeksiantajana Health Tech Lab siis halusi oman sovelluksen, joka keräisi laitteilla saadun datan yhteen paikkaan ilman, että tarvitsisi käyttää Healthmatea. Health Tech Labin toivoma lopputulos oli ohjelmakoodi, joka kerää ja päivittää Withings-laitteilla saadun datan automaattisesti, sekä näiden laitteiden

yleinen käyttöönotto, sekä laitteiden ja niihin liittyvän API:n toiminnan selvittäminen. Näitä laitteita ei siis ennen tämän opinnäytetyön aloittamista ollut käytetty lainkaan, eikä niiden toimintaan tai API:n toimintaan ollut perehdytty myöskään.

Tässä opinnäytetyössä käydään läpi käytännön kehitystyön vaiheet yksityiskohtaisesti. Tämä läpikäynti on eritelty työssä pääsäännöllisesti käytettyjen kolmen teknologian mukaisesti, eli Withings API, Node-Red ja React. Ymmärtämisen ja seuraamisen helpottamiseksi mukana on useita kuvia työn tärkeimmistä kohdista. Työssä perehdytään myös yleisesti API:en toimintaan käyttötapoihin.

2 API JA TIEDONSIIRTO

2.1 API

2.1.1 Mikä on API

API on lyhenne sanoista Application Programming Interface, eli sovellusohjelmointirajapinta. API mahdollistaa sovelluksen kommunikoinnin toisen sovelluksen kanssa ja tiedon siirtämisen automaattisesti yhdistettyjen sovellusten kesken, ilman että kehittäjän tarvitsee tietää tarkkaan, miten kyseinen toiminto on toteutettu. Esimerkiksi kehittäjä voi Google Maps API:n avulla sisältää Google Maps -objektin verkkosivulleen, ilman että hänen tarvitsee tietää yksityiskohtaisesti, miten Google Mapsin -tyylinen karttasovellus toimii. Ilman API:n apua kehittäjä joutuisi itse kehittämään oman karttasovelluksensa ja tuottaa itse kaikki tarvittava data tätä varten, joka olisi lähes mahdotonta tai tuhlaisi aivan liikaa aikaa sen tuottamaa arvoa kohden. API:t siis voivat yksinkertaistaa sovelluksen kehitysprosessia ja säästävät aikaa ja rahaa. [2, 8, 15, 26]

Ensimmäiset API:t kehitettiin tietojenkäsittelyn todella aikaisessa vaiheessa, jo ennen kotikoneiden yleistymistä. Tällöin API:t olivat lähes aina paikallisia kirjastoja käyttöjärjestelmissä. API:en käyttö etäisesti tiedon välityksessä yleistyi vasta 2000-luvun alussa. [15]

Tällaisista etäkäyttöisistä API:sta, joiden muokkaamat resurssit ovat jossain muualla kuin paikallisesti tietokoneella, käytetään nimitystä Remote API, eli etäinen API. Web API eli verkko API on eräänlainen etäinen API, joka on suunniteltu internet standardien mukaiseksi. Esimerkki verkko API:sta on Twitterin API, joka antaa esimerkiksi mahdollisuuden etsiä twiittejä, jotka sisältävät jotain tiettyä, kehittää uusia Twitter mainoksia ja saada tietoa tilien aktiivisuudesta. Verkko API:t käyttävät yleensä http-tyyppisiä pyyntöviestejä ja antavat vastauksen näihin pyyntöihin joko XML- tai JSON-tiedoston muodossa. XML- ja JSON-muotoja suositaan vastauksen muotona, koska niiden tyyppiset tiedostot ovat helppoja muokata muille sovelluksille ja ohjelmille. [2, 15]

2.1.2 API:en käyttö ja hyöty

Tiedonsiirron lisäksi API:a voi käyttää myös tietoturvaan liittyvissä asioissa. API:n avulla voidaan kontrolloida muiden pääsyä tiettyihin tietoihin tai osiin ohjelmasta, joihin heillä ei muuten olisi pääsyoikeutta. Tästä esimerkki on mikä tahansa sovellus tai verkkosivusto, joka pyytää pääsyä käyttäjän sijaintiin. Tällöin selain tai sovellus käyttää hyväkseen API:a saadakseen tietoonsa käyttäjän sijainnin helposti, ilman, että sivuston tai ohjelman kehittäjän tarvitsee itse kehittää jonkinlainen monimutkainen ohjelma, joka tekisi saman asian. Kun verkkosivu pyytää käyttäjän sijaintitietoja näin API:n kautta, on käyttäjällä mahdollisuus myös olla antamatta tähän lupaa, jos hän ei sitä halua. Toinen positiivinen asia käyttäjälle tässä on, että jos hän suostuu antamaan sijaintinsa, hän voi olla varma, että API antaa sovellukselle tai sivustolle pääsyn vain ja ainoastaan hänen sijaintitietoihinsa, eikä mihinkään muuhun salattuun tietoon, ilman hänen suostumustaan. [2]

Toinen tietoturvaan liittyvä asia on OAuth-standardi, joka määrittää useaa API:a, ja joka antaa mahdollisuuden kirjautua tiettyihin sivustoihin käyttäen jonkin muun sivuston, kuten esimerkiksi Googlen-, Facebookin- tai Twitterin-tunnuksia. OAuthin avulla käyttäjä voi siis kirjautua sivustolle, luomatta uusia käyttäjätunnuksia, jonkin toisen sivuston tunnuksillaan. Hänen ei tällöin myöskään kuitenkaan luovuta tälle uudelle sivustolle esmierkiksi Google-tilinsä salasanaa. OAuth ei siis anna toiselle sivustolle pääsyä käyttäjän koko Google-tilille, vaan se antaa pääsyn tiettyihin osiin hänen tiedoistaan, kuten nimeen ja sähköpostiosoitteeseen. Tällöin kun käyttäjä yrittää kirjautua uudestaan tälle sivustolle Google-tunnustensa kanssa, Google hoitaa todennusprosessin toisen sivuston sijaan. [1,2]

API:t mahdollistavat myös teknologia yritysten integroinnin toistensa kanssa. API:n tarjoaja voi sen avulla antaa muille pääsyn omiin resursseihinsa, antamatta myös sen hallintaa muille. API:en julkaisussa muille on kolme eri tapaa, joista jokainen tarjoaa eri etuja. Näistä ensimmäinen on yksityinen julkaisu, joka tarkoittaa, että API on käytössä vain sen luoman yrityksen sisäisesti. Tämä tapa tarjoaa sen luojalle eniten hallintaa sen luomusta kohti. Toinen tapa on kumppani julkaisu. Tämä tarkoittaa, että API:iin annetaan käyttöoikeus vain tietyille kumppani yrityksille ja entiteeteille. Tämä antaa API:n luoneelle yritykselle takuun, että API:n käyttö ja käyttötarkoitus ovat asianmukaisia, sekä se saattaa tuottaa yritykselle uusia tulonlähteitä. Kolmas julkaisutapa on yleinen julkaisu, joka tarkoittaa, että API on käytössä kenelle tahansa. Yritys luovuttaa tällöin pääsyoikeuden hallinnan API:iin, mutta se saattaa samalla olla hyvä tapa edistää innovointia kun

mahdollisimman monella on oikeus käyttää sitä. Edistynyt innovaatio taas saattaa johtaa yrityksen brändin leviämiseen uusiin paikkoihin, uusien tulonlähteiden syntymiseen tai se voi parantaa tehokkuutta yrityksen sisällä ulkoisen kehityksen ja yhteistyön kautta. [15, 22]

2.1.3 SOAP ja REST

Verkko API:en yleistyessä niiden tiedon siirtoa on pyritty standardoimaan ja näistä tunnetuimmat ja käytetyimmät protokollat ovat SOAP ja REST. SOAP on lyhenne sanoista Simple Object Access Protocol ja REST taas sanoista Representational State Transfer. Yleensä API on joko SOAP:in tai REST:in mukainen, ja sen valinta riippuu usein API:in käyttötarkoituksesta ja kehittäjän mieltymyksestä. [14, 15]

SOAP on virallinen protokolla, jota ylläpitää World Wide Web Consortium tai W3C. Se suunniteltiin alun perin mahdollistamaan eri ohjelmointikielillä tai alustoilla luotujen ohjelmien kommunikoinnin. SOAP:iin perustuvat API:t käyttävät datan kyselyssä lähes mitä tahansa tiedon muotoa kuten http tai SMTP, mutta vastaukset se antaa XML-muodossa. Tämä ei kuitenkaan ole niin hyödyllistä kuin voisi kuvitella, sillä nykyään lähes kaikki verkkopohjainen toiminta toimii http:n kautta. Koska SOAP on virallinen protokolla, siihen perustuvien API:en kehittämiseen liittyy useita sääntöjä ja ohjeita, joita tulee noudattaa ja jotka tekevät niistä yleensä hitaampia ja jäykempiä kuin REST:iin perustuvat API:t. Kaikki SOAP:in tarjoamat toiminnot ovat myös todella tarkasti määritelty ja standardoitu. SOAP on yleisesti parempi vaihtoehto, jos yhdistetään useita eri sovelluksia, jotka on luotu eri ohjelmointikielillä tai alustoilla.[14, 18, 24]

REST taas ei SOAP:in mukaisesti ole standardi vaan arkkitehtuurinen tyyli. Tämä tarkoittaa, että REST:iin perustuvien API:en kehitykselle ei ole mitään yhtenäistä standardia, vaan se antaa kehittäjälle ohjeet ja antaa kehittäjän toteuttaa ne omalla tavallaan. REST perusteisesta API:sta käytetään nimitystä RESTful API. RESTful API:t eivät siis ole tyyliltään ja toteutukseltaan yhtä jäykkiä ja hankalakäyttöisiä kuten SOAP:iin perustuvat API:t. RESTful API:ssa datan kyselyssä käytetään lähes aina http-muotoa ja vastaus voi olla muotoa HTML, XML, JSON tai tavallinen teksti. JSON on kuitenkin näistä yleisin. REST onkin yleisesti SOAP:ia tehokkaampi, koska se mahdollistaa pienempien data formaattien kuten JSON:in käytön, kun taas SOAP käyttää ainoastaan XML:aa. REST on myös nopeampi ja kevyempi käyttää kuin SOAP ja tämän takia se soveltuukin

paremmin käyttöön pienemmissä sovelluksissa, kuten mobiilisovelluksissa. Esimerkiksi Google Maps API on RESTful API, kuten myös Withings API. [14, 18]

API:sta voidaan sanoa sen olevan RESTful jos se täyttää kuusi tiettyä määritettä. Näistä ensimmäinen on asiakas - palvelin arkkitehtuuri. Tällä tarkoitetaan, että API:n arkkitehtuuriin kuuluu asiakas, palvelin ja resurssit, joita ne lähettelevät keskenään. Seuraava määritelmä on tilattomuus (statelessness), jolla tarkoitetaan, että mitään asiakkaan tietoja ei säilytetä palvelimella, kun minkäänlaisia pyyntöjä ei olla tekemässä. Jokainen erillinen pyyntö palvelimelle sisältää kaiken tarvittavan asiakkaan tiedon, jotta pyyntö saadaan suoritettua. Kolmas määritelmä on välimuistillisuus (cacheability). Tämä tarkoittaa, että joitain asiakkaan tietoja voidaan säilyttää välimuistissa, jotta voidaan vähentää asiakkaan ja palvelimen välisiä toimenpiteitä. Neljäntenä on kerrostettu järjestelmä (layered system), joka tarkoittaa, että asiakkaan ja palvelimen kanssakäymisen voi hoitaa joku toinen ohjelma, mutta sen pitää olla huomaamatonta asiakkaalle. Tämä voi antaa API:lle muun muassa lisättyä tietoturvaa. Seuraava osa on koodi tarvittaessa (code on demand), ja se on ainoa vaihtoehtoinen määritelmä. Tämä tarkoittaa, että palvelin voi väliaikaisesti antaa asiakkaalle uusia toimintoja tai muokata olemassa olevia, lähettämällä hänelle toteuttamiskelpoisia koodin osia. Viimeinen määritelmä on yhtenäinen rajapinta, joka tarkoittaa, että tietoa siirretään asiakkaan ja palvelimen välillä yhtenäisellä tavalla. [14, 23]

REST on siis nykyään suositumpi vaihtoehto kuin SOAP, sen antaman vapauden ja nopeamman toiminnan takia. REST antaa kehittäjälle vapaat kädet toteuttaa sen määritykset hänen haluamallaan tavalla, kun taas SOAP antaa tarkat ohjeet, joita tulee noudattaa juuri niiden mukaan.

3 TOTEUTUS

3.1 Withings API

3.1.1 Mikä on Withings API

Withings API on Withingsin kehittämä API, jonka avulla kehittäjät voivat luoda ohjelmia, jotka käyttävät hyväkseen Withings laitteita ja niiden tallentamaa dataa. API:n avulla kehittäjillä on pääsy HealthMate sovelluksen säilyttämään dataan, jonka he voivat integroida omiin toimiinsa, olettaen, että he saavat luvan datan käyttöön sen omistajalta. API on julkinen, eli sitä voi käyttää kuka tahansa ja sen pyyntöjen maksimimäärä on 120 pyyntöä minuutissa. [27]

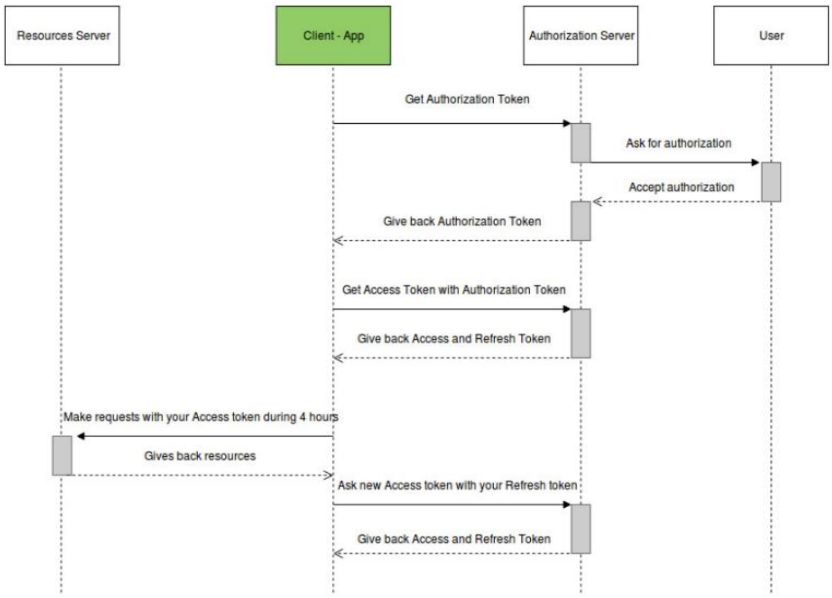
API:lla voi saada käyttöönsä kaikkea dataa, mikä tallentuu Health Mate -sovellukseen kaikista Withingsin laitteista, jotka ovat kyseiseen Withings-profiiliin liitetty. Tämä tarkoittaa niin profiilin luomisen yhteydessä syötettyä dataa, kuten pituutta ja painoa, kuin myös esimerkiksi tässä opinnäytetyössä käytettyjä tietoja, kuten veren ylä- ja alapainetta, ruumiinlämpöä, REM-unen pituutta ja monia muita.

3.1.2 Withings API:n käyttöönotto

Ennen kuin API:a voi käyttää, tulee käyttäjän ensin luoda Withings-profiili, joka on sama profiili, jolla kirjaudutaan sisään Health Mate -sovellukseen. Käyttäjän on myös luotava Withings-kehittäjäsovelluksen sen verkkosivuilla. Tämä vaatii muun muassa logon, toimivan sähköpostiosoitteen ja callback URI:n, joka on vapaasti valittava verkkosivu, johon käyttäjä uudelleenohjataan todennusprosessin aikana. Kehittäjä saa myös tältä verkkosivulta koodin, joka vaaditaan käyttäjän tietoihin käsiksi pääsyyn. Kehittäjäsovelluksen luominen antaa käyttäjälle hänen asiakas ID:nsä ja niin kutsutun kuluttaja salaisuuden, jotka ovat todella tärkeitä, sillä niitä käytetään lähes kaikissa API-kutsuissa. [27]

3.1.3 Todennusprosessi ja OAuth 2.0

Tämän jälkeen käyttäjän tulee suorittaa todennusprosessi ja tätä varten Withings API käyttää OAuth 2.0 -protokollaa. OAuth 2.0 on avoin standardi, jonka kautta käyttäjä voi antaa kolmannen osapuolen ohjelmalle pääsyn tiettyyn osaan heidän tiedoistaan, paljastamatta salasanaansa. OAuth 2.0 -prosessi toimii niin, että ensin kolmas osapuoli pyytää käyttäjältä käyttöoikeutta heidän tiettyihin tietoihinsa kyseisessä ohjelmassa tai sivustolla, johon käyttäjä voi suostua tai ei. Jos käyttäjä suostuu tähän, saa kolmas osapuoli jonkin tyyppisen myöntymyksen, jonka avulla se voi saada niin sanotun käyttölupatunnuksen valtuutuspalvelimelta. Tämän käyttölupatunnuksen avulla se taas pystyy pyytämään sille valtuutettuja tietoja kyseessä olevalta ohjelmalta tai sivustolta (Kuva 1). OAuth 2.0 protokollaa käyttävät myös esimerkiksi Twitter, Facebook ja Google. [4, 11, 21]



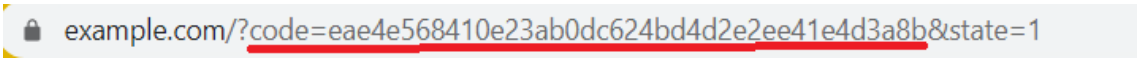
Kuva 1. OAuth 2.0 prosessi Withings API:n yhteydessä. [27].

Withings API:ssä tämä prosessi toimii hyvin samantyyppisesti. Ensin käyttäjän, jolta pyydetään pääsyä hänen tietoihinsa, tulee siirtyä seuraavalle verkkosivulle:

https://account.withings.com/oauth2_user/authorize2?response_type=code&redirect_uri=[REDIRECT_URI]&scope=[SCOPE]&state=1&client_id=[CLIENT_ID]

Dataan pääsyä pyytävän tulee asettaa kehittäjäsovelluksen luonnin yhteydessä saatu asiakas ID kohtaan [CLIENT_ID], hänen itse aikaisemmin päätetyn callback URI:n kohtaan [REDIRECT_URI] ja kohdan [SCOPE] tilalle tulee asettaa se, mihin osaan käyttäjän datasta halutaan pääsyoikeus. Tähän kohtaan vaihtoehdot ovat user.info, joka antaa pääsyn käyttäjän perustietoihin, user.metrics, joka antaa pääsyn käyttäjän mittausdataan, user.activity, joka antaa pääsyn käyttäjän aktiviteetti dataan ja user.sleepevents, joka antaa pääsyn käyttäjän uni tapahtumiin. Kehittäjä voi pyytää samalla pääsyä yhteen tai useampaan osaan käyttäjän datasta, jolloin eri laajuudet pitää erotella pilkulla. [6] **(tästä kaikesta joku kuva/prosessikaavio)**

Kun käyttäjä navigoi kyseiselle sivulle, hänen pitää kirjautua Withings-tililleen, minkä jälkeen häneltä kysytään, haluaako hän antaa luvan hänen datansa käytölle. Kun käyttäjä hyväksyy tämän, hän siirtyy sivulle, jonka kehittäjä asetti sovelluksensa callback URI:ksi. Siellä kehittäjän on saatava sivun URL palkissa oleva koodi, joka toimii OAuth 2.0:n vaatimana myöntymyksenä (Kuva 2). [6]



Kuva 2. Todennusprosessiin tarvittava koodi URL palkissa.

3.1.4 Käyttölupatunnuksen saaminen

Tämä koodi on voimassa vain 30 sekuntia, jonka aikana kehittäjän on käytettävä koodia joko Curl tai PHP komennossa saadakseen käyttölupatunnuksen käyttäjän tietoihin. Curl on komentokehote työkalu ja kirjasto, jota käytetään datan siirtämisessä paikasta toiseen URL syntaksia käyttäen. [3] PHP taas on ohjelmointikieli, jota käytetään erityisesti dynaamisten verkkosivujen luomisessa. [12] Tämän opinnäytetyön tekemiseen käytettiin Curlia, joten tässä osiossa olevat komennot ovat Curl-komentoja.

Saadakseen käyttölupatunnuksen, kehittäjän tulee syöttää seuraava Curl-komento komentokehotteeseensa:

curl --data "grant_type=authorization_code&client_id=[CLIENT_ID]&client_secret=[CLIENT_SECRET]&code=[CODE]&redirect_uri=[CALLBACK_URI]" https://account.withings.com/oauth2/token

Samoin kuin aiemmin kehittäjän tulee laittaa kohtiin [CLIENT_ID] ja [CONSUMER_SECRET] hänen saamansa asiakas id ja kuluttaja salaisuus ja [CALLBACK_URI] kohdan tilalle itse päätetty callback URI. Tämän lisäksi kohtaan [CODE] tulee laittaa edellisen kohdan URL palkista saatu koodi. Tämä komento pitää syöttää 30 sekunnin sisällä koodin saamisesta, koska koodi on vain sen aikaa voimassa. Jos prosessissa menee kauemmin kuin 30 sekuntia, tulee todennus prosessi aloittaa alusta.

Kun komennon saa syötettyä ajoissa, kehittäjä saa vastauksen, joka näyttää tältä:

{

"access_token": "string",

"expires_in": 0,

"token_type": "string",

"scope": "string",

"refresh_token": "string",

"userid": "string"

}

Vastauksen muotoilu ei aina välttämättä näytä samalta, mutta sen sisältö on silti sama. Tässä vastauksessa tärkeät osat ovat access_token, eli käyttölupatunnus, sen voimassaolo aika ja refresh_token, eli päivitystunnus. Käyttölupatunnusta käytetään kaikissa Curl komennoissa, joilla haetaan käyttäjän generoimaa dataa. Käyttölupatunnus on aina voimassa kolme tuntia, jonka jälkeen voi käyttää saatua päivitystunnusta sen uusimiseen. Käyttölupa- ja päivitystunnus kannattaa siis aina ottaa muistiin johonkin.

Käyttölupatunnuksen vanhennuttua kehittäjä voi syöttää seuraavan komennon Curliin saadakseen uuden tunnuksen:

curl --data "grant_type=refresh_token&client_id=[CLIENT_ID]&client_secret=[CONSUMER_SECRET]&refresh_token=[REFRESH_TOKEN]" https://account.withings.com/oauth2/token

Tämä komento toimii samoin kuin alkuperäinen komento, jolla saatiin Käyttölupatunnus. [CLIENT_ID] ja [CONSUMER_SECRET] kohtine tilalle pitää asettaa oikeat arvot ja

[REFRESH_TOKEN] kohtaan alkuperäisen käyttölupatunnuksen yhteydessä saatu päivitystunnus. Tämän komennon antama vastaus on täysin samanlainen, kuin komennon, jolla alun perin saatiin käyttölupatunnus. Se antaa uuden käyttölupatunnuksen, joka on myös voimassa seuraavat kolme tuntia ja myös uuden päivitystunnuksen, jota tulee käyttää seuraavan kerran, kun valtuutus pitää uusia. Pitää myös ottaa huomioon, että myös päivitystunnuksella on voimassaoloaika. Withingsin dokumentaatiosta ei löydy tarkkaa aikaa tälle, mutta API:n kanssa työskentelyn perusteella arvioisin sen olevan noin seitsemän päivää. Jos käy niin, että myös päivitystunnus ehtii vanhentua, kehittäjän pitää aloittaa alusta todennusprosessista lähtien.

3.1.5 Datan hakeminen palvelimelta

Kun kehittäjällä on voimassa oleva käyttölupatunnus, hän voi sen avulla hakea dataa, johon datan omistaja antoi käyttöluvan. Tämän opinnäytetyön yhteydessä käytetyt laitteet olivat BPM Core verenpainemittari, Sleep unenseurantamatto ja Thermo älykuumemittari. Käytetyillä komennoilla haettiin siis vain niillä saatua dataa. Ensimmäinen käytetty komento oli:

curl -H "Authorization: Bearer [ACCESS_TOKEN]" https://wbsapi.withings.net/measure?action=getmeas&meastype=9,10,11,12,71,73&category=1&lastupdate=[LAST_UPDATE]&offset=XX

Tällä komennolla saatiin kaikki verenpaine- ja kuumemittarin tuottama data. Tähän komentoon piti lisätä [ACCESS_TOKEN] kohtaan voimassa oleva käyttölupatunnus ja kohtaan [LAST_UPDATE] päivämäärä, jonka jälkeen luotua dataa haluttiin. Tämä päivämäärä pitää olla UNIX-aika muodossa, joka on ajan tallennustapa, joka kertoo ajan sekunteina ajanhetkestä 1. tammikuuta 1970 kello 00:00:00 UTC. [25] Esimerkiksi nämä kyseiset laitteet otettiin käyttöön 27.11.2019, joten käytettiin sitä päivämäärää [LAST_UPDATE] kohdassa, joka on UNIX-aikana 1574830000. Näiden lisäksi tärkein osa komennossa on kohta meastype, jolle on annettu arvot 9, 10, 11, 12, 71 ja 73. Tämä kohta kertoo tarkasti mitä tiettyä dataa halutaan ja jokainen numero merkitsee jotain tiettyä osaa kokonaisdatasta. Numero 9 on verenpaineen alapaine, numero 10 on verenpaineen yläpaine, numero 11 on sydämen syke, numero 12 on lämpötila, numero 71 on ruumiinlämpö ja numero 73 on ihon lämpö. [27]

Toinen käytetty komento, jolla saatiin kaikki unimaton tuottama data, oli:

curl -H "Authorization: Bearer [ACCESS_TOKEN]" https://wbsapi.withings.net/v2/sleep?action=getsummary&lastupdate=[LAST_UPDATE]&data_fields=breathing_disturbances_intensity,deepsleepduration,durationtosleep,durationtowakeup,hr_average,hr_max,hr_min,lightsleepduration,remsleepduration,rr_average,rr_max,rr_min,sleep_score,snoring,snoringepisodecount,wakeupcount,wakeupduration

Tämä komento on hyvin samanlainen kuin edellinenkin. [ACCESS_TOKEN] ja [LAST_UPDATE] kohtiin menevät samat arvot kuin edellisessä komennossa. Ainoa merkittävä ero näiden kahden komennon välillä on, että jälkimmäisessä meastype-kohdan tilalla on kohta data_fields. Nimestä huolimatta, ne toimivat lähes identtisesti, paitsi unidataa pyydettäessä ei käytetä numeroita tietyn osan hakuun vaan itse datanosan nimeä. Esimerkiksi jos halutaan saada keskimääräinen syke yön aikana, käytettäisiin arvoa hr_average.

3.2 Saadun datan käsittely ja Node-Red

3.2.1 Mikä on Node-Red

Kun Withings API:n saa toimimaan ja sillä saa haettua käyttäjän tuottamaa dataa, tarvittiin tapa, jolla siihen saa yhteyden ohjelmointi työkalun kanssa, jotta sen saisi esille rajapintaan. Tätä varten käytimme ohjelmaa nimeltään Node-Red.

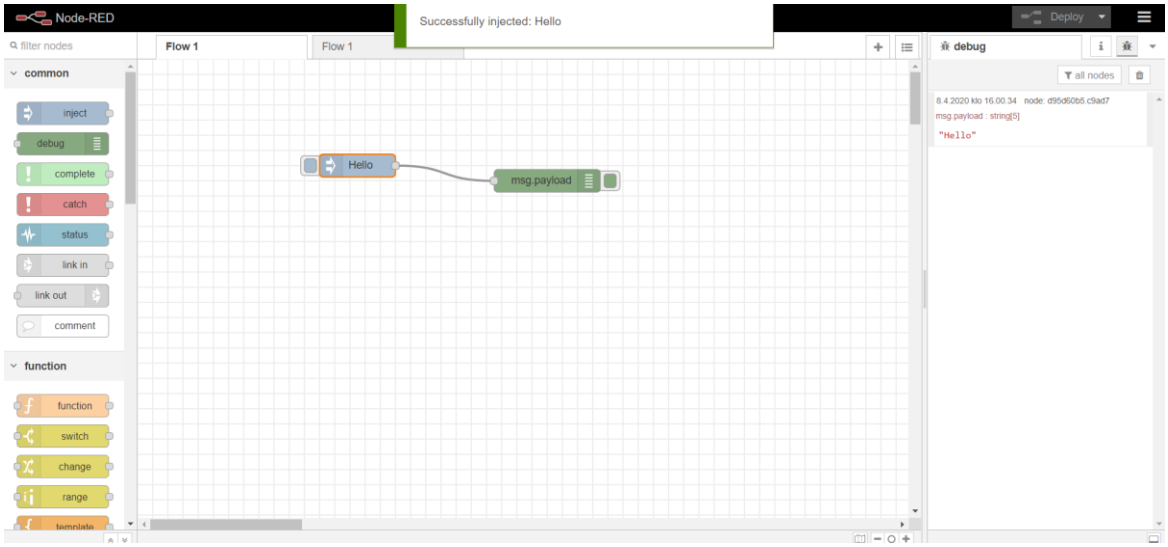
Node-Red on Node.js -pohjainen visuaalinen ohjelmointi työkalu, jota käytetään muun muassa API:den ja muiden verkkopalveluiden toisiinsa yhdistämisessä. Tämä tapahtuu Node-Redin visuaalisen selaimessa toimivan rajapinnan avulla, jossa matalatasoiset ohjelmointi tehtävät on korvattu niin kutsutuilla solmuilla (node), jotka kehittäjä voi vetää ja tiputtaa muokkaimeen ja yhdistää toisiinsa. Jokaisella solmulla on ennalta ohjelmoitu tehtävä, jonka se suorittaa, joten itse ohjelmointia ei yleensä tarvita, vaikka onkin myös mahdollista muuttaa solmun toimintaa muokkaamalla sen ohjelmakoodia. Kun solmuja yhdistää toisiinsa, syntyy niin kutsuttu vuo (flow). Node-Redin avulla luotiin siis yksinkertainen vuo, johon luotu rajapinta voi ottaa yhteyden ja esittää Withings laitteilla saatu data sivulla. [10, 17]

3.2.2 Node-Red vuon luominen

Ennen Node-Redin käyttöa, kehittäjän pitää ladata itse Node-Red. Node-Redin voi ladata usein eri tavoin ja useille eri laitteille. Sen voi ladata muun muassa paikallisesti omalle tietokoneelle käyttäen npm komentorivi työkalua, Raspberry Pi:lle, joka on yhden piirilevyn tietokone, tai vaikka Android laitteelle. Tätä opinnäytetyötä varten latasin Node-Redin lokaalisti npm työkalua käyttäen. [10]

Kun Node-Red on ladattu paikallisesti, ohjelman voi käynnistää avaamalla komentokehotteen ja antamalla siihen komennon node-red. Tämän jälkeen Node-Red muokkaimen voi avata verkkoselaimessa menemällä osoitteeseen *http://localhost:1880*, jos käytät selainta samalla tietokoneella, joka pyörittää Node-Redia tai jos käytössä on selain toisella tietokoneella, on käytettävä Node-Redia pyörittävän tietokoneen IP-osoitetta ja mentävä osoitteeseen *http://<ip-osoite>:1880*. Jos Node-Redia käyttää näin paikallisesti, on myös komentokehotteen, jolla se käynnistettiin oltava auki jatkuvasti, tai ohjelma sammuu. Kun ohjelma on käynnissä voi aloittaa solmujen asettelun muokkaimeen ja niiden yhdistämisen toisiinsa. Kaikki tehdyt muutokset tallentuvat automaattisesti. [10]

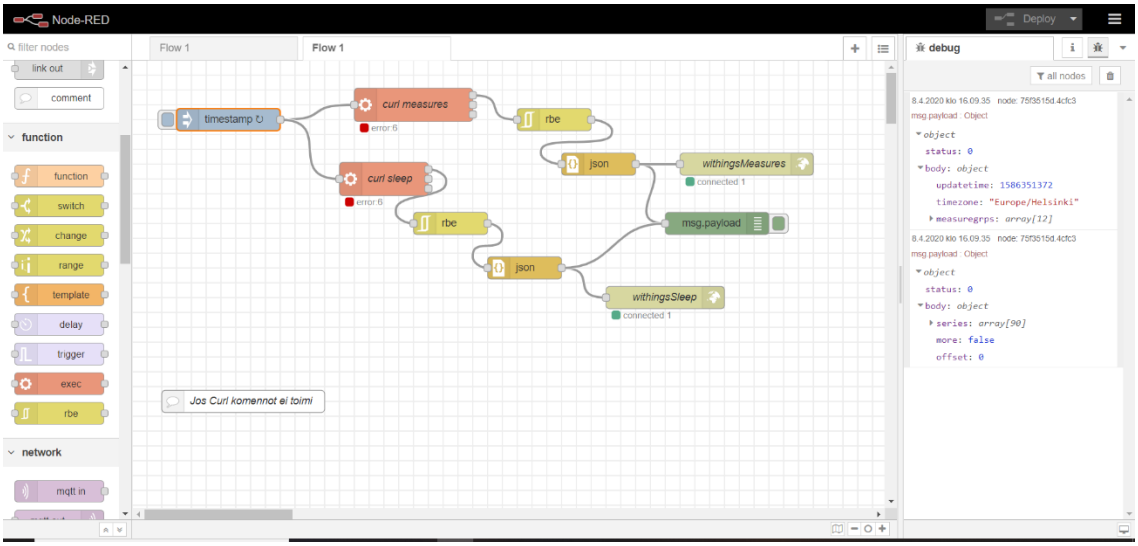
Lähes kaikki Node-Red vuot sisältävät ainakin niin kutsutun inject- eli injektiosolmun, joka lisää jonkinlaisen viestin vuohon, joko tietyin aikavälein tai manuaalisesti. Tästä viestistä käytetään nimitystä payload eli tietosisältö. Tietosisältö voi olla useaa eri tyyppiä, esimerkiksi merkkijono, Javascript-objekti tai nykyinen aika. Toinen lähes jokaisessa vuossa oleva solmu on debug- eli testaussolmu, joka esittää injektiosolmun lähettämän tietosisällön Node-Redin oikean reunan debug-ikkunassa (Kuva 3).



Kuva 3. Yksinkertainen vuo, jossa on injektiosolmu, testaussolmu ja lähetetty viesti debug ikkunassa.

3.2.3 Luodun vuon esittely

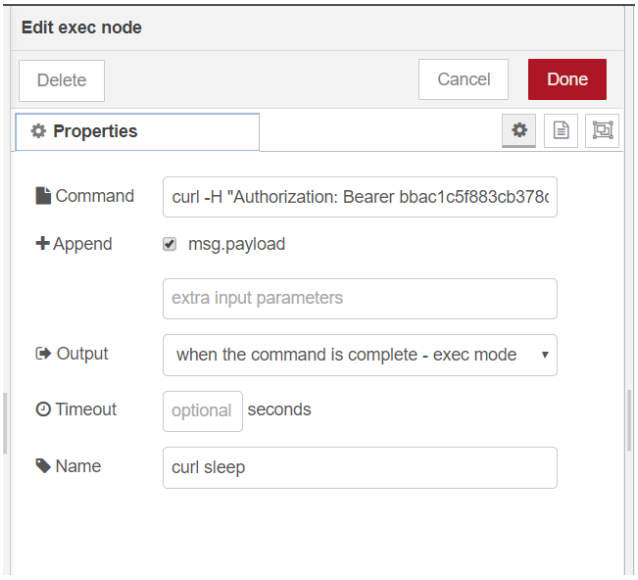
Tätä opinnäytetyötä varten luotu vuo on melko yksinkertainen. Sen tehtävä on hakea kappaleessa 3.1.5 esitellyillä Curl komennoilla käyttäjän data, jonka jälkeen se muutetaan JSON muotoon ja tehdään mahdolliseksi siihen yhdistäminen luodun rajapinnan ohjelmakoodissa (Kuva 4).



Kuva 4. Opinnäytetyön valmis lopullinen vuo ja sen tulostama viesti.

Kaikki alkaa sinisestä injektiosolmusta, joka lisää vuohon tietosisällön, joka on tässä muotoa timestamp, eli sen hetkinen aika. Solmu lähettää uuden tietosisällön joka 60 sekunnin välein, eli vuo toistaa toimintaansa jatkuvasti. Tämä on sen takia, että jos Health Tech Labin Withings laitteilla generoidaan uutta dataa, ohjelma hakee sen saman tien ja se lisätään rajapintaan näkyviin.

Tästä injektiosolmusta erkanee kaksi identtistä tietosisältöä, jotka lähdettyään siirtyvät kahteen eri niin kutsuttuun punaiseen exec-solmuun, nimeltään curl sleep ja curl measures. Vuo haarautuu tässä vaiheessa, koska molemmat puolet tulevat saamaan eri arvot ja niillä on eri päämäärät. Exec-solmun tarkoitus on ajaa sille annettu komentorivikomento ja palauttaa sen vastaus (Kuva 5). Näillä solmuilla haettiin käyttäjän data kappaleessa 3.1.5 esitellyillä komennoilla. Curl measures solmulle annettiin komento, joka hakee verenpaine- ja kuumemittarin datan ja curl sleep solmulle annettiin komento, joka hakee unimaton tuottaman datan. Kun tietosisältö kulkee näiden solmujen läpi, niiden sisällöt muuttuvat sisältämään unimaton datan ja verenpaine- ja kuumemittarien datan vastaavasti.

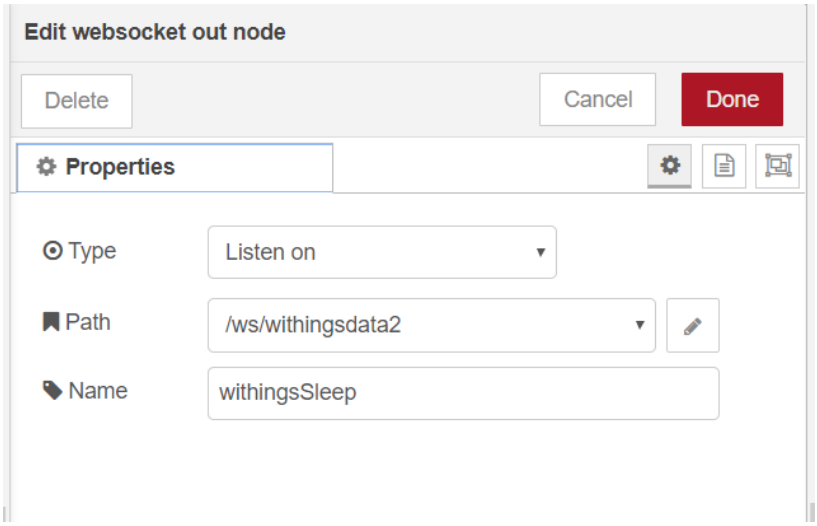


Kuva 5. Exec-solmu, jossa sille annettu komento kohdassa Command.

Tämän jälkeen tietosisältö etenee vaalean keltaiseen niin kutsuttuun rbe-solmuun. Rbe on lyhenne sanoista Report By Exception eli ilmoita poikkeuksesta. Tämän solmun tarkoitus on estää tietosisällön eteneminen, jos sen sisältö ei ole muuttunut edellisestä kerrasta, kun se kulki solmun läpi. Tämä solmu lisättiin sen takia, ettei Node-Redin debug-ikkuna sotkeentuisi turhaan ja ettei rajapinta jatkuvasti lataisi sisältöään uudestaan

vaikka sen saama data ei olisi muuttunut lainkaan. Tästä seuraava solmu on JSON-solmu, väriltään tumman keltainen, jonka tarkoitus on yksinkertaisesti vain muuttaa sen saama tietosisältö JSON dataformaattiin ja lähettää se eteenpäin.

Tämän jälkeen vuo haarautuu jälleen, kun molemmat haarat kulkevat yhteiseen vihreään testaussolmuun, joka esittää molempien haarojen viestit debug-ikkunassa. Tämä solmu on tässä sen takia, että voi tarkastaa, että niiden sisältö on oikeanlainen. JSON-solmusta lähtevä toinen haara taas menee beigen väriseen niin kutsuttuun websocket out-solmuun. Websocket on protokolla, joka mahdollistaa jatkuvan yhteyden palvelimen ja asiakkaan välillä, antaen niiden siirtää dataa toisesta toiseen. Tämän tyyppisestä yhteydestä käytetään nimitystä TCP, eli Transmission Control Protocol, yhteys. [7] Tässä projektissa palvelimena toimii tämä Node-Red vuo ja asiakkaana toimii rajapinta, jossa Withings laitteiden data on esillä ja tämä yhteys näiden välillä saadaan yhdistämällä näihin kahteen websocket out-solmuun (Kuva 6). WithingsSleep niminen



Kuva 6. Websocket out-solmu, jossa kohdan Path arvoon yhdistetään kun luodaan yhteys rajapinnan kanssa.

solmu lähettää nimensä mukaisesti unimaton dataa ja withingsMeasures niminen solmu verenpaine- ja kuumemittarin dataa.

Injektiosolmun alapuolella oleva valkoinen solmu on kommenttisolmu, joka kertoo mitä pitää tehdä, jos vuo ei toimi. Tällöin ongelmana on todennäköisesti Curl komentojen vanhentuminen, jolloin kommentti kertoo, miten tulee menetellä. Tämä prosessi on sama, kuin luvussa 3.1.4 kuvattu uuden käyttölupatunnuksen saaminen.

3.3 Rajapinnan luominen ja React

3.3.1 Mikä on React

Node-Red vuon luonnin jälkeen tarvitaan jokin rajapinta, jolla siihen otetaan yhteys. Tällaisen rajapinnan kehittämiseen valittiin React tai React.js niminen ohjelmointikehikko. React on JavaScript kirjasto, jota käytetään käyttäjärajapintojen kehittämiseen yksisivuisille websovelluksille tai mobiilisovelluksille. Reactin päätarkoitus on olla mahdollisimman nopea ja yksinkertainen. Sen avulla kehittäjä voi luoda suuriakin websovelluksia, jotka voivat muuttaa sisältöään ilman sivun uudelleen latausta. Tämän takia React sopii hyvin API:en kanssa työskentelyyn ja tähän kyseiseen opinnäytetyöhön, koska kun Health Tech Labin Withings laitteilla tuotetaan uutta dataa, se tulee suoraan näkyviin rajapintaan ilman ohjelman uudelleen latausta. [9, 13]

Reactin käyttöön suositeltu ohjelmointikieli on nimeltään JSX. JSX on lyhenne sanoista JavaScript XML ja se on eräänlainen laajennus normaaliin JavaScriptin syntaksiin. JSX:n avulla voi käyttää HTML komentoja ja tunnisteita JavaScriptin sisällä, jolloin ne muuttuvat niin kutsutuiksi React elementeiksi. JSX:n käyttö ei kuitenkaan ole pakollista, sillä Reactin kanssa voi käyttää myös normaalia JavaScriptia. [13, 20]

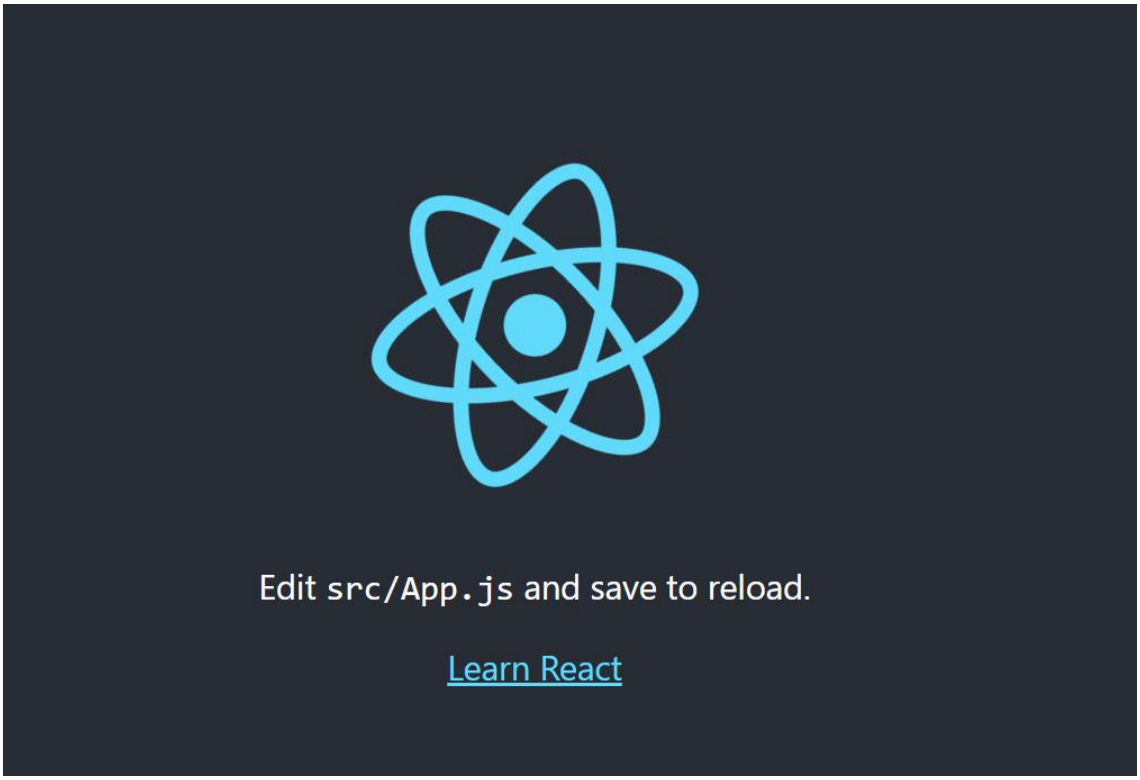
Tässä projektissa Reactia käytettiin siis luomaan rajapinta, joka yhdistää Node-Red vuon websocketteihin ja esittää niistä saadun datan sivulla. Rajapinta myös päivittyy automaattisesti, jos websockettien tietosisällössä on jotain uutta dataa.

3.3.2 React ohjelman luominen

React kirjaston asentamiseen tietokoneelle on useita eri tapoja, mutta helpoin tapa on käyttää Create-React-App nimistä pakettia. Tämän paketti tulee ensin ladata käyttäen npm komentorivi työkalua syöttämällä komentokehotteeseen komennon *npm install -g create-react-app*. Tämä komento lataa Create-React-App paketin, jonka jälkeen kehittäjä voi luoda React sovelluksen. Sovelluksen luominen tapahtuu komennolla *npx create-react-app <sovelluksen nimi>*, jossa sovelluksen nimi kohtaan annetaan sovellukselle haluttu nimi. Tätä komentoa käyttäessä pitää myös ottaa huomioon, missä tiedostossa tai kansiossa on tällä hetkellä komentokehotteessa, sillä *create-react-app*

komento luo sovelluksen siihen kansioon, jossa kehittäjä on sillä hetkellä. Kehittäjä voi siirtyä kansiosta toiseen käyttämällä *cd <kansion nimi>* komentoa. [16]

Kun sovellus on luotu, se avataan ensin navigoimalla sille kuuluvaan kansioon komennolla *cd <sovelluksen nimi>*, ja sen jälkeen käynnistämällä sovelluksen komennolla *npm start*. Tämän jälkeen sovellus on käynnissä paikallisesti osoitteessa *http://localhost:3000* tai osoitteessa *http://<ip-osoite>:3000,* jos siihen haluaa yhdistää toiselta tietokoneelta (Kuva 7). [16]



Kuva 7. Muokkaamaton React sovellus.

Kun sovellus on toiminnassa, jotta sen sisältöä pääsee muokkaamaan, pitää sen ohjelmakoodi avata oma valintaisessa koodinmuokkausohjelmassa. Tässä opinnäytetyössä käytettiin Visual Studio Code ohjelmointiympäristöä. Visual Studio Codessa halutun ohjelman ohjelmakoodin voi avata avaamalla vasemman yläkulman File valikon, valitsemalla sieltä vaihtoehdon Open File ja valitsemalla sen jälkeen ohjelman, jonka ohjelmakoodin haluaa avata. Tämän jälkeen itse sovelluksen kehittämisen voi aloittaa muokkaamalla ohjelmakoodia. Kun Reactissa tekee muutoksia ohjelmakoodiin ja sen tallentaa, jos sovellus on auki selaimessa, se päivittyy reaaliaikaisesti uusilla muutoksilla, ilman että sivua pitäisi itse päivittää manuaalisesti.

3.3.3 Node-Rediin yhdistäminen

React rajapinnan ja Node-Red vuossa olevan websocketin yhdistäminen toimii kuvassa 6 näkyvän polun avulla. Tämän polun arvo annetaan itse nimetylle muuttujalle, jonka jälkeen käytetään *onopen()* nimistä metodia, joka yhdistää rajapinnan websockettiin. Kun näiden yhdistäminen onnistuu ohjelma antaa viestin rajapinnan konsoliin, sanoen että yhdistäminen onnistui (Kuva 8).

```
class App extends Component {
  constructor() {
    super();
    this.state = {
      endpoint1: "ws://localhost:1880/ws/withingsdata",
      endpoint2: "ws://localhost:1880/ws/withingsdata2",
      measuresData: [],
      uniData: []
    };
  }

  componentDidMount() {
    const ws1 = new WebSocket(this.state.endpoint1);
    ws1.onopen = () => {
      console.log("ws1 connected");
      //connects to websocket1
    };
```

Kuva 8. Yhdistetään websockettiin.

Kun yhdistäminen on onnistunut käytetään metodia nimeltään *onmessage()*, joka toimii kun websocketeista saadaan uusi viesti. Kun viesti saapuu, erotellaan siitä ensin hyödyllinen ja hyödytön data ja sen jälkeen se muutetaan array eli taulukkomuotoon. Data muutetaan taulukoksi, jotta se voidaan eritellä datan keräämispäivämäärän mukaisesti, jolloin se näyttää siistimmältä sivulle tulostettuna. Lopuksi saatu taulukko asetetaan itse keksityn muuttujan arvoksi, jolloin se voidaan tulostaa. Tämä prosessi tehdään kummankin websocketin viestille, jolloin saadaan kaksi erillistä taulukkoa, toinen, joka sisältää

unimaton tuottaman datan ja toinen, joka sisältää verenpaine- ja kuumemittarin tuottaman datan (Kuva 9).

```
ws1.onmessage = evt => {
  const kayttajaDataMeasures = JSON.parse(evt.data);
  console.log(kayttajaDataMeasures);
  //parses the data and prints it to console

  const filteredData = kayttajaDataMeasures.body.measuregrps;
  const kayttajaDataMeasuresArray = Object.values(filteredData);
  //filters out the unnescessary data and changes the rest into an array

  this.setState({
    measuresData: kayttajaDataMeasuresArray
    //sets the array to state
  });
  console.log(this.state.measuresData);
};
```

Kuva 9. Muotoillaan websocketista saatu data oikeean muotoon.

3.3.4 Komponentit

Kun uuden React sovelluksen avaa koodinmuokkausohjelmassa, pitäisi siinä olla näkyvissä kansiot node_modules, public ja src ja kansiosta src pitäisi löytyä tiedosto nimeltään App.js. Tämä App.js tiedosto kerää kirjoitetun ohjelmakoodin ja lähettää sen eteenpäin index.js nimiselle tiedostolle, joka puolestaan tulkitsee tämän koodin ja kuvaa sen tuottamat asiat sovelluksen rajapintaan. Halutessaan kehittäjä voi kirjoittaa kaiken ohjelmakoodinsa App.js tiedostoon ja tämä onkin ihan sopivaa pienissä ja yksinkertaisissa sovelluksissa, mutta jos koodia syntyy paljon, kannattaa se jakaa komponentteihin.

Reactissa komponentit ovat itsenäisiä, uudelleenkäytettäviä koodin osia, jotka toimivat samalla tavalla kuin JavaScript funktiot, mutta jonkin arvon palauttamisen sijaan ne palauttavat HTML-elementin, eli osan verkkosivua. Esimerkiksi tämän opinnäytetyön React sovellus on jaettu App.js -tiedoston lisäksi viiteen eri komponenttiin, joista jokainen tekee ja palauttaa tietyn osan siitä, mitä sovellus näyttää sivulla, kun sen käynnistää.

Ensimmäinen ja yksinkertaisin komponentti on nimeltään navbar.jsx ja se ei sisällä paljonkaan koodia ja sen tarkoitus on vain palauttaa rajapinnan yläosan tumma palkki ja siinä oleva kuva ja teksti. Palkki näkyy kuvassa 11.

Tämän sovelluksen App.js -tiedosto sisältää kappaleessa 3.3.3 kuvatun Node-Rediin yhdistämisen ja saatujen viestien muokkaamisen käytettävään muotoon. Tämän lisäksi siihen kuulu vielä näiden muokattujen viestien lähettäminen omiin komponentteihinsa, jotka puolestaan hoitavat datan lopullisen muokkaamisen ja näytölle asettamisen. Kun Reactissa lähetetään jonkinlaista dataa niin sanotusti alaspäin komponenteille, siitä käytetään nimitystä props. Ne komponentit, jotka vastaanottavat tämän datan, ovat nimeltään sleepdata.jsx, coredata.jsx ja thermodata.jsx ja nimensä mukaisesti niistä jokainen käsittelee yhden tietyn Withings-laitteen tuottaman datan.

Nämä komponentit ovat myös melko yksinkertaisia. Koska verenpaine- ja lämpömittarin tuottama data saapuu kaikki samassa taulukossa, pitää coredata ja thermodata komponenteissa ensin eritellä pois toisen laitteen data. Tämä tapahtuu kappaleessa 3.1.5 selitettyjen eri datatyyppiä kuvaavien numeroiden avulla. Sleepdata komponentissa tällaista erittelyä ei tarvitse tehdä, koska sen mukana ei tule mitään ylimääräistä. Kun erottelu on saatu kuntoon ja jokaisella komponentilla on vain niille hyödyllinen data, käytetään map() nimistä funktiota, joka ryhmittelee ja asettaa jo ennestään taulukko muodossa olleen datan listaksi, jokaisella taulukon kohdalla olevan id arvon mukaan ja päivämäärällisesti vanhimmasta uusimpaan, eli kun laitteilla dokumentoidaan uutta dataa se lisätään automaattisesti jokaisen listan pohjalle (Kuva 10).

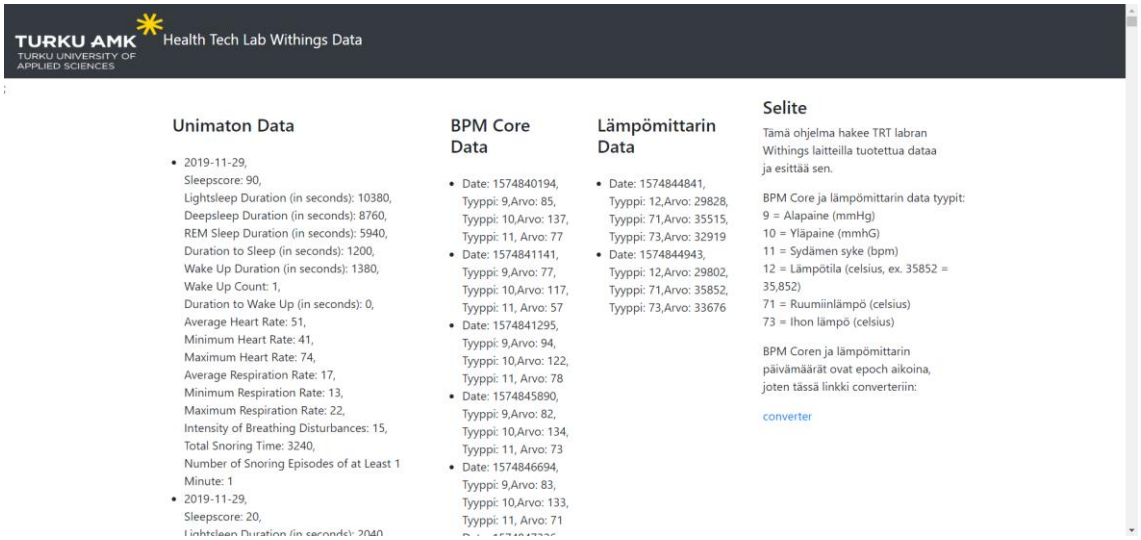
```
src > components > coredata.jsx > ...
1   import React, { Component } from "react";
2
3   class CoreData extends Component {
4     state = {};
5
6     render() {
7       let dataWithDeviceID = this.props.saatuData.filter(d => {
8         return d.deviceid !== null;
9         //filters out the user data
10      });
11
12      let noThermoData = dataWithDeviceID.filter(d => {
13        return d.measures[0].type === 9 && 10 && 11;
14        //filters out the thermometer data
15      });
16
17      return (
18        <div>
19          <h4 className="m-4">BPM Core Data</h4>
20          <ul>
21            {noThermoData.map(d => (
22              <li key={d.grpid}>
23                {"Date: " + d.date},<br /> {"Tyyppi: " + d.measures[0].type},
24                {"Arvo: " + d.measures[0].value},<br />{" "}
25                {"Tyyppi: " + d.measures[1].type},{"Arvo: " + d.measures[1].value}
26                ,<br /> {"Tyyppi: " + d.measures[2].type},{" "}
27                {"Arvo: " + d.measures[2].value}
28              </li>
29            ))}
30          </ul>
31        </div>
32      );
33      //maps the remaining data into a list
34    }
35  }
36
37  export default CoreData;
```

Kuva 10. coredata.jsx komponentin koodi.

Viimeinen tehty komponentti on nimeltään legend.jsx eli selite, joka sisältää vain yhden tekstikentän, joka kertoo rajapinnan käyttötarkoituksen, kertoo mitä verenpaine- ja lämpömittari osioissa olevat numerot tarkoittavat ja koska myös näiden kahden osion päivämäärät ovat luvussa 3.1.5 selitetyssä UNIX-aikamuodossa annetaan myös linkki internetsivulle, missä tämän ajan voi muuttaa ymmärrettävämpään muotoon. Yhdessä nämä viisi komponenttia kokoavat rajapinnan näkymän.

3.3.5 Lopputulos

React osuuden lopputuloksena, oli siis rajapinta, joka automaattisesti tulostaa näytölle TRT laboratorion Withings laitteilla tuotetun datan ja päivittää kyseistä dataa jatkuvasti (Kuva 11).



Kuva 11. Lopullinen versio rajapinnasta.

4 YHTEENVETO

Opinnäytetyön tavoitteena oli luoda Health Tech Labille rajapinta, jossa laboratorion omistamien Withings-laitteiden tuottamat tiedot olisivat näkyvillä ja joka päivittäisi tietoja automaattisesti. Tällainen rajapinta saatiin toteutettua ja se saatiin toimimaan halutusti, mutta jotain parannettavaakin jäi. Esimerkiksi Withings API:ssa olevat rajoitukset johtivat siihen, että vaikka rajapinta periaatteessa päivittää dataa automaattisesti, tarvitsee se silti ylläpitoa Withings API:n käyttölupatunnuksen asettamisessa ja uusimisessa. Tästä kerrotaan tarkemmin luvussa 4.1.

Ajallisesti opinnäytetyössä kesti hieman odotettua kauemmin, ainakin omasta mielestäni, vaikka työlle ei asetettu mitään konkreettista aikarajaa. Opinnäytetyö aloitettiin marraskuun lopulla, jolloin Withings-laitteet otettiin käyttöön ensimmäistä kertaa ja rajapinta valmistui maaliskuun puolessa välissä. Työn etenemistä viivästytti muun muassa käytettyjen teknologioiden tuntemattomuus. En ollut ennen tätä opinnäytetyötä käyttänyt lainkaan Reactia, Node-Redia tai Withings APi:ta, joten niihin totuttelussa kului melko reilusti aikaa. Tulin kuitenkin työn aikana tutuksi kaikkien näiden ohjelmien kanssa ja opin myös käyttämään niitä melko hyvin, mikä on positiivinen asia. Niissä on kaikissa myös tietysti paljon toimintoja, joita en päässyt tässä työssä käyttämään, joten niissä on silti vielä opittavaakin.

Uskoisin myös, että opinnäytetyöstä on hyötyä myös toimeksiantajalle. Sen tekemisessä saatiin ainakin paljon uutta tietoa Withings laitteiden toiminnasta ja siitä, miten niiden dataan voi päästä käsiksi ja miten sitä voi hyödyntää. Myös Node-Red vuo ja React rajapinta toteuttavat tarkoituksensa, vaikkakin eivät aivan täydellisesti. Silti uskoisin, että niissä on ainakin paljon potentiaalia jatkokehitykseen, jolloin niiden mahdolliset viat voidaan korjata ja voidaan mahdollisesti löytää parempi tapa projektin toteutukseen, jolla tietyt ongelmat saataisiin korjattua.

4.1 Kohdatut ongelmat

Kaiken kaikkiaan Withings API toimii melko hyvin, kun siihen tottuu ja sitä alkaa ymmärtämään. Siinä on kuitenkin useita ongelmia, joihin minä ja muut sitä käyttäneet ovat törmänneet. Ensinnäkin sen formatointi on usein epäjohdonmukaista ja sekavaa. Esimerkiksi lähes jokaisessa tiedonhakukutsussa tarvitaan jonkinlainen päivämäärä, mutta

tämän tarvittavan päivämäärän formaatti vaihtelee näennäisesti kutsusta kutsuun. Osassa niistä käytetään edellä mainittua UNIX-aikaformaattia, kun taas toisissa pyydetään YMD eli vuosi, kuukausi, päivä -formaatin mukaista päivämäärää. Tähän päivämääräformaattiin liittyvät vaikeudet eivät myöskään loppuneet tähän, vaan myös tämän YMD-muodon kanssa oli ongelmia. API:n dokumentaatio ei kerro, miten tällainen päivämäärä tulisi muotoilla ja ainoastaan pitkän testaamisen jälkeen totesin, että se tulee olla muotoa \[27/Nov/2019\], joka ei edes ole muotoa vuosi, kuukausi, päivä vaan päinvastoin. [5]

Sanoisin myös, että API:n käyttäjäystävällisyys on hieman epätyydyttävää. Ohjelma ei muun muassa aina kerro, mikä on minkäkin haetun datan osion arvon yksikkö. Esimerkiksi API:n dokumentoinnissa kerrotaan, että unimaton antama durationtosleep eli nukahtamiseen kulunut aika on sekunneissa, mutta saman laitteen snoring eli kuorsauksen kesto osiossa ei ole kerrottu, onko arvo sekunneissa, minuuteissa vai tunneissa. Myös se, että ohjelma antaa useat arvot sekunteina, on mielestäni ongelma, sillä jos haetaan esimerkiksi syvän unen pituutta, saa arvon sekunteina, vaikka kyseinen pituus voi hyvinkin olla monta tuntia. Eli jos kyseisen yön syvän unen pituus oli kolme tuntia, antaa API arvon 10 800 sekuntia, joka käyttäjän pitää itse muuttaa tunneiksi tai minuuteiksi, jolloin se on helpompi hahmottaa.

Withings API:n dokumentaatio jättää myös joissain osissa toivomisen varaa. Varsinkin OAuth 2.0 ja todennusosioissa, jossa dokumentaatio oli niin epäselvää, että siihen piti hakea apua ulkopuolisilta sivuilta. Samassa osiossa dokumentaatio myös väittää käyttölupatunnuksen olevan voimassa neljä tuntia, vaikka todellisuudessa se on voimassa vain kolme tuntia.

Muita ongelmia, joihin en itse ole tämän työn aikana törmännyt mutta joita muut käyttäjät ovat kohdanneet, ovat muun muassa, että API:lla ei voi hakea kaikkea dataa, jota tietty laite tuottaa. Ilmeisesti API ei esimerkiksi voi hakea Withings Smart Body Analyzerin tuottamaa ilman laatu arvoa. [5]

Node-Red-työkalu on melko yksinkertainen, ja sitä voivat käyttää nekin, joilla ei ole hyviä ohjelmointitaitoja. Työkalun ongelmana on, että on vaikea selvittää sen toimintoja, esim. miten työssäni tarvitsema vuo tehdään. Koska tämä työkalu ei ole kovin tunnettu, ei siitä ole julkaistu tutoriaaleja, oppaita tai muita tukityökaluja kehittäjän avuksi. Node-Redin sivuilla on kuitenkin hyvä aloitustutoriaali, mutta sitä vaativampia tutoriaaleja ei ole.

Toinen merkittävä ongelma oli Withings API:n toiminnan kanssa. Tarkemmin sanottuna koska Withingsin API:n tunnustautumisprosessi on monimutkainen, sitä ei pystynyt automatisoimaan Node-Redin kanssa. Tämän takia kehittäjän on manuaalisesti tehtävä käyttölupatunnuksen hakeminen ja uusiminen aina sen vanhennuttua ja laitettava uusi tunnus Exec-solmujen Curl-komennoissa olevien vanhentuneiden tunnusten tilalle, jotta koko projekti toimii. Uusimisprosessi on luvussa 3.1.4 kuvattu prosessi. Tämä tuottaa lisää ylläpitotyötä, mutta parempaa vaihtoehtoa tälle ei löytynyt.

Reactissa törmättyjen ongelmien syy taas oli lähes aina vain ohjelmointikielen osaamisen puute ja muutenkin ohjelmointiympäristön tuntemattomuus. Kun tutustuin Reactiin enemmän, alkoi sen käyttö myös sujua paremmin. Kuitenkin myös tähän osioon jäi joitain ongelmia, joita en saanut korjattua ja joista kerrotaan enemmän luvussa 4.2.

4.2 Jatkokehitysideoita

Lopulliseen projektiin jäi siis joitain asioita, joita en saanut korjattua, joihin ei yksinkertaisesti löytynyt parempaa vaihtoehtoa tai jotka pystyttäisiin mahdollisesti korjaamaan jatkokehityksessä. Näistä ensimmäinen ja merkittävin on luvussa 4.1 kuvailtu Node-Red -vuossa oleva ongelma Withings API:n tunnustautumisprosessin kanssa. Alun perin Node-Redin oli tarkoitus tehdä koko datan hakemisprosessi automaattisesti ja jatkuvasti. Pitkän tutkimuksen ja testaamisen jälkeen ei kuitenkaan löytynyt tapaa tehdä sitä näin, joten päädyimme toimeksiantajan kanssa tulokseen, että tämä tunnistautumisprosessi pitää tehdä manuaalisesti, kun ohjelmaa haluaa käyttää, mutta käyttölupatunnus on vanhentunut.

Toinen jatkokehityksessä mahdollisesti parannettava asia on React-osiossa. Kuten kuvasta 11 näkyy, rajapinnan oikeassa reunassa on osio nimeltään selite, joka kertoo, mitä verenpaine- ja lämpömittariosioissa olevat numerot merkitsevät ja se antaa myös linkin verkkosivulle, jossa käyttäjä voi muuttaa samoissa osioissa olevat UNIX-ajat ymmärrettävämpään muotoon. Tämä selitys ja arvojen muuttaminen oli alun perin tarkoitus tehdä suoraan näille arvoille, eli UNIX-ajan tilalla olisi automaattisesti normaali päivämäärä ja aika ja esimerkiksi numeron 9 tilalla lukisi verenpaineen alapaine, numeron 10 kohdalla lukisi verenpaineen yläpaine ja niin edelleen. Tämän tekeminen ei kuitenkaan onnistunut halutulla tavalla, jolloin päätettiin tehdä rajapinnalle tekstikenttä, jossa nämä vain selitettäisiin. Selitteen oli siis tarkoitus olla osana rajapintaa vain tilapäisesti, ja sen voi poistaa helposti, jos jatkokehityksessä tämän toiminnon saa kuntoon.

Samantyyppinen hienosäätöasia on unimaton dataosioissa, jossa yksi nukuttu yö unimaton päällä tuottaa paljon dataa, joka puolestaan vie paljon tilaa näytöllä. Tällä hetkellä tämä data ilmestyy näytölle kronologisessa järjestyksessä, eli uusin data ilmestyy jonon pohjalle. Ja koska yhden yön data vie paljon tilaa, jos haluaa nähdä vaikkapa edellisen yön unidatan, joutuu rajapinnassa vierittämään näyttöä todella kauas alaspäin. Tämän takia yksi jatkokehitysidea olisi, että unidata esitettäisiin käänteiskronologisessa järjestyksessä, eli uusin ensin. Tämän lisäksi rajapinta voisi esittää unidataa vain esimerkiksi kymmeneltä viimeisimmältä päivältä ja vain käyttäjän pyynnöstä se lataisi enemmän dataa näytölle. Tämä parantaisi rajapinnan käytettävyyttä merkittävästi ja vähentäisi lataamisaikaa, koska ohjelman ei tarvitse ladata niin paljon dataa kerralla. Tätä toimintoa voisi myös käyttää verenpaine- ja kuumemittarin datojen osioissa, jos niillä kerätään merkittävästi dataa. Tällä hetkellä tämä ei vielä kuitenkaan ole pakollista.

LÄHTEET

[1] Chris Hoffman. 2017. What Is OAuth? How Those Facebook, Twitter, and Google Sign-in Buttons Work. Viitattu 11.5.2020. Saatavilla: https://www.howtogeek.com/53275/exchanging-data-safely-with-oauth/

[2] Chris Hoffman. 2018. What Is an API?. Viitattu 23.4.2020. Saatavilla: https://www.howto-geek.com/343877/what-is-an-api/

[3] Curl. [Viitattu: 3.4.2020] Saatavilla: https://curl.haxx.se/

[4] D.Hardt. 2012. The OAuth 2.0 Authorization Framework. Viitattu 3.4.2020. Saatavilla: https://tools.ietf.org/html/rfc6749

[5] Katemonkeys. 2017. Everything Wrong With The Withings API Viitattu 6.4.2020. Saatavilla: https://gist.github.com/katemonkeys/e17580777b57915f5068

[6] Kodymallory. 2018. MMM-Withings. Viitattu 3.4.2020. Saatavilla: https://github.com/kodymal-lory/MMM-withings

[7] MDN contributors. 2020. Websockets. Viitattu 9.4.2020. Saatavilla: https://develo-per.mozilla.org/en-US/docs/Glossary/WebSockets

[8] Mulesoft. What is an API? (Application Programming Interface). Viitattu 23.4.2020. Saatavilla: https://www.mulesoft.com/resources/api/what-is-an-api

[9] Nitin Pnadit. 2020. What and Why React.js. Viitattu 15.4.2020. Saatavilla: https://www.c-sharp-corner.com/article/what-and-why-reactjs/

[10] Node-Red. Viitattu 10.4.2020. Saatavilla: https://nodered.org/

[11] OAuth 2.0. Viitattu 2.4.2020. Saatavilla: https://oauth.net/2/

[12] PHP. Viitattu 1.4.2020. Saatavilla: https://www.php.net/

[13] React. Viitattu 17.4.2020. Saatavilla: https://reactjs.org/

[14] Red Hat. REST vs. SOAP. Viitattu 11.5.2020. Saatavilla: https://www.red-hat.com/en/topics/integration/whats-the-difference-between-soap-rest

[15] Red Hat. What is an API?. Viitattu 23.4.2020. Saatavilla: https://www.red-hat.com/en/topics/api/what-are-application-programming-interfaces

[16] Ross Chesley. 2019. How to Install React JS On Windows. Viitattu 15.4.2020. Saatavilla: https://www.liquidweb.com/kb/install-react-js-windows/

[17] Shashidhar Soppin. 2017. Everything you need to know about Node-RED. Viitattu 9.4.2020. Saatavilla: https://opensourceforu.com/2017/09/node-red/

[18] SoapUI. SOAP vs REST 101: Understand The Differences. Viitattu 11.5.2020. Saatavilla: https://www.soapui.org/learn/api/soap-vs-rest-api/

[19] Turku AMK. Health Tech. Viitattu 23.4.2020. Saatavilla: https://healthtech.turkuamk.fi/

[20] w3Schools.com. React JSX. Viitattu 16.4.2020. Saatavilla: https://www.w3schools.com/react/react_jsx.asp

[21] Whitson Gordon. 2012. Understanding OAuth: What Happens When You Log Into a Site with Google, Twitter, or Facebook. Viitattu 3.4.2020. Saatavilla: https://lifehacker.com/understanding-oauth-what-happens-when-you-log-into-a-s-5918086

[22] Wikipedia. Application programming interface. Viitattu 11.5.2020. Saatavilla: https://en.wikipedia.org/wiki/Application_programming_interface

[23] Wikipedia. Representational State Transfer. Viitattu 11.5.2020. Saatavilla: https://en.wikipedia.org/wiki/Representational_state_transfer

[24] Wikipedia. SOAP. Viitattu 11.5.2020. Saatavilla: https://en.wikipedia.org/wiki/SOAP

[25] Wikipedia. UNIX-aika. Viitattu 1.4.2020. aatavilla: https://fi.wikipedia.org/wiki/UNIX-aika

[26] Visma. API – Mikä on API?. Viitattu 22.4.2020. Saatavilla: https://www.visma.fi/epasseli/kirjanpidon-sanakirja/a/api/

[27] Withings API developer documentation (v1.0). 2019. Viitattu 17.4.2020. Saatavilla: https://developer.withings.com/oauth2/#tag/changelog

[28] Withings. Viitattu 23.4.2020. Saatavilla: https://www.withings.com/fi/en/

[29] Withings. Suunniteltu parempaan terveyteen. Viitattu 20.4.2020. Saatavilla: https://www.withings.com/fi/fi/welcome-fi