Tampereen yliopisto

Matti Suorsa

# ÄLYKODIN IOT-JÄRJESTELMÄN KEHITTÄMINEN

## Firebase IoT-alustana

Informaatioteknologian ja viestinnän tiedekunta
Pro gradu -tutkielma
Maaliskuu 2021

# TIIVISTELMÄ

Matti Suorsa : Älykodin IoT-järjestelmän kehittäminen - Firebase IoT-alustana
Pro gradu -tutkielma
Tampereen yliopisto
Tietojenkäsittelyopin maisteriopinnot
Maaliskuu 2021

---

IoT-järjestelmän reunalla ovat IoT-laitteet, kuten tietoa keräävät sensorit sekä toimenpiteitä tekevät aktuaattorit. Sensorit ja aktuaattorit eivät usein kykene kommunikoimaan itsenäisesti suoraan Internetissä, ja niiden keräämä data usein vaatii muuntamista tai esiprosessointia ennen kuin sitä voidaan käyttää. Tähän tehtävään sovelletaan IoT-yhdyskäytäviä. Yhdyskäytävän täytyy pystyä kommunikoimaan laitteiden sekä itse IoT-alustan kanssa, jonne tieto lopulta siirretään. Tieto säilötään IoT-alustalla tietokannassa, jossa sitä voidaan myös prosessoida ja analysoida. Alustalle kerätty tieto visualisoidaan sekä esitetään reaaliaikaisesti käyttöliittymässä, josta voidaan myös ohjata järjestelmän laitteita.

Kokonainen IoT-järjestelmä rakentuu useista tasoista, joiden välille vaaditaan asianmukaiset protokollat tiedon siirtämiseen sekä keinot tiedon säilömiseen. Järjestelmän kehittämisessä tulee huomioida myös, kuinka saadaan säilytettyä yhteentoimivuus järjestelmän tasojen ohjelmistojen välillä ja kuinka sen skaalautuvuus voidaan varmistaa.

Työn tavoitteena oli kehittää yksinkertainen älykodin IoT-järjestelmä. Järjestelmälle luotiin oma IoT-yhdyskäytävä Raspberry Pi:llä, ja sen ohjelmisto kehitettiin Python-ohjelmointikielellä. Kommunikointiin yhdyskäytävän ja laitteiden välillä käytettiin HTTP-pohjaista RESTful API:a. IoT-alustaksi sovellettiin Google Firebase -pilvipalvelua, jossa tieto tallennettiin NoSQL-tietokantaan. Firebasen palveluiden käyttäminen toteutettiin sille tarkoitettujen ohjelmistokehityspakettien avulla. Järjestelmän käyttöliittymä luotiin Vue.js-ohjelmistokehyksellä, ja käyttöliittymässä tietoa visualisoitiin Chart.js JavaScript-kirjastoa apuna käyttäen.

Pilvipalvelun ilmainen maksusuunnitelma saatiin riittämään käyttötarkoitukseen ja käytetyt frameworkit pohjautuivat avoimen lähdekoodiin, joten kaikki järjestelmän kehittämiseen vaadittavat kulut muodostuivat käytettävästä laitteistosta. Firebasen tarjoamat ominaisuudet soveltuivat sen käyttämiseen IoT-alustana, eikä erilliselle palvelimelle ollut tarvetta. Firebasea voidaan siis erityisesti käyttää IoT-järjestelmän prototyypin kehittämisessä, ja sen skaalautuvuuden sekä tehokkuuden vuoksi myös todellisissa projekteissa, ainakin osana järjestelmää.

Avainsanat: IoT, Firebase, Cloud, Pilvipalvelu, Raspberry Pi, Vue.js

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

# ABSTRACT

Matti Suorsa : The development of smart home IoT System - Firebase as an IoT-Platform
Master of science thesis
Tampere University
Computer Science master's degree
March 2021

---

At the edge of an IoT-system there are the IoT-devices, such as sensors that collect information, and actuators that do operations. Sensors and actuators can't usually communicate with the Internet directly, and the data they have collected usually needs preprocessing or transforming before it can be used. IoT-gateways can be used for these tasks. Gateway device must support communication with IoT-devices and the IoT-platform where the data will eventually be transferred. The data is stored in a database of an IoT-platform, where also the data can be processed and analyzed. The data collected to the platform is visualized and displayed in real time on a user interface. These interfaces can also be used to control the IoT-devices.

A complete IoT-system consists of multiple layers that require the use of appropriate data transfer protocols and means for storing data. When developing an IoT-system it is required to consider how to retain the interoperability between the softwares of each layer, and how the scalability in the system can be ensured.

The goal of this study was to develop a simple IoT-system for smart home purposes. IoT-gateway was built with Raspberry Pi for the IoT-system, and its software was developed with Python programming language. HTTP-based RESTful API was used for the communication between the gateway and the IoT-devices. In the IoT-system, Google Firebase cloud service was used as an IoT-platform, where also the data was stored in a NoSQL-database. The use of Firebase services was done with the help of its software development kits. The user interface of the system was built with Vue.js JavaScript framework and the data was visualized in the interface with the help of Chart.js JavaScript library.

The free payment plan of the cloud service was suitable enough for the purpose and the frameworks that were used were open-source so all the costs required for the development of the system were based on the hardware used. The features offered by Firebase were suitable for using it as an IoT-platform and there was no need for a separate server. Firebase can especially be used in the development of an IoT-system prototype, and because of it's scalability and performance, it can also be used in a real project, atleast as a part of one.

Keywords: IoT, Firebase, Cloud, Cloud service, Raspberry Pi, Vue.js

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

# SISÄLLYSLUETTELO

1 JOHDANTO ..... 1
2 MENETELMÄT ..... 2
2.1 IoT ..... 2
2.2 IoT-laitteet ..... 2
2.3 IoT-järjestelmän arkkitehtuuri ..... 4
2.4 IoT-protokollat ..... 5
2.5 Yhteentoimivuus ..... 6
2.6 Web-teknologiat ja käsitteet ..... 8
2.7 Tietokannat ..... 9
2.8 IoT-alustat ..... 9
3 IOT-LAITEET ..... 12
3.1 Raspberry Pi ..... 12
3.2 Kehittäminen ..... 13
3.3 Philips Hue ja RESTful API ..... 16
3.4 Yhteenveto ..... 18
4 PILVIALUSTA - FIREBASE ..... 19
4.1 Firebase console ..... 19
4.2 Tietokannat ..... 21
4.2.1 Cloud Firestore ..... 21
4.2.2 Ohjelmistokehityspaketit ..... 23
4.2.3 Kehittäminen ..... 23
4.2.4 Turvallisuussäännöt ja käyttäjän todentaminen ..... 26
4.3 Pilvifunktiot ..... 27
4.4 Yhteenveto ..... 27
5 KÄYTTÖLIITTYMÄ JA TIEDON VISUALISOINTI ..... 28
5.1 Firebase CLI & Firestore Web SDK ..... 29
5.2 Vue.js ..... 30
5.2.1 Kehittäminen ..... 32
5.2.2 Visualisointi ..... 33
5.3 Yhteenveto ..... 34
6 OMA TUTKIMUS JA TULOKSET ..... 36
6.1 Järjestelmän kuvaus ..... 36
6.1.1 IoT-laitteet ..... 37
6.1.2 IoT-alusta ..... 40
6.1.3 Käyttöliittymä ..... 42
6.1.4 Testit ..... 45
6.2 Lopputulokset ..... 46

6.3 Jatkokehitys ........................................ 47
7 LÄHDELUETTELO ........................................ 49

# LYHENTEET

| | |
|---|---|
| **IoT** | Internet of Things eli Esineiden Internet |
| **IIoT** | Industrial Internet of Things eli teollinen Esineiden Internet |
| **NoSQL** | Ei-relaatiomalliin pohjautuva tietokanta |
| **SDK** | Ohjelmistokehityspaketti |
| **API** | Ohjelmointirajapinta |
| **REST** | Arkkitehtuurinen tyylisuunta yhteentoimivuuden parantamiseen |
| **RPI** | Raspberry Pi, yhden piirilevyn tietokone |
| **CLI** | Komentoliittymä |
| **DOM** | Rajapinta HTML- ja XML-dokumenttien manipulointiin |
| **GPIO** | Yleiskäyttöinen Input/Output-pinni mikrokontrollereissa |
| **JSON** | Ihmisen luettava tiedonvälitykseen tarkoitettu tiedostomuoto |
| **NPM** | Node-paketinhallintajärjestelmä, engl. Node Package Manager |
| **GUI** | Graafinen käyttöliittymä |

# 1 JOHDANTO

IoT-alustalla on keskeinen asema IoT-järjestelmässä. Oman alustan kehittäminen sekä ylläpito vaativat runsaasti resursseja, joten valmiin alustan käyttäminen on usein tarpeellista. IoT-alustaksi tarjolla on useita valmiita ratkaisuja aina omalle palvelimelle asennettavista alustoista pilvipalveluiden tarjoamiin pilvialustoihin. Alustan valitsemisessa tulee ottaa huomioon, miten tietoa voidaan siirtää, säilöä ja käsitellä alustalla, kuinka suuren osan ajasta tiedon tulee olla saatavilla, alustan hinta ja turvallisuus sekä sen mahdollistamat kehitystyökalut. Usein todelliset, projektikohtaiset tarpeet selviävätkin vasta kehitysvaiheessa, joten IoT-järjestelmästä on järkevää kehittää prototyyppi.

Tässä työssä käydään läpi älykodin yksinkertaisen IoT-järjestelmän luominen sovelluskehityksen merkeissä. Tutkimuksessa sovelletaan Google Firebase -pilvipalvelua IoT-alustana, Raspberry Pi:tä IoT-laitteena sekä Vue.js-ohjelmistokehystä web-käyttöliittymän kehittämiseen. Pilvipalvelun tarjoama ojelmistokehityspaketti on olennaisessa osassa kehitystä, mutta myös perinteiset web-sovelluskehityksen teknologiat ja käytännöt ovat myös tärkeässä asemassa.

Työ siis keskittyy pääasiassa IoT-järjestelmän web-puoleen, eikä niinkään laitteistoon tai laitteiden väliseen kommunikaatioon. Työ ei myöskään käsittele oman alustan kehittämistä, mutta työstä voi saada hyvän käsityksen siitä, mitä ominaisuuksia alustan tulee sisältää. On hyvä ottaa huomioon, että omassa tutkimuksessa toteuttamani ratkaisu ei todennäköisesti ole toimiva ainakaan suuren järjestelmän kehittämisessä; kyseessä on nimensä mukaisesti tutkimus valituista teknologioista sekä niiden toimivuudesta IoT-käytössä. Koska IoT-järjestelmään sisältyy valtava määrä teknologioita, niin paljon oleellistakin jää varmasti käsittelemättä.

Menetelmät osassa, eli toisessa luvussa käydään läpi tutkimuksen kannalta oleellisimmat aiheet ja teknologiat sekä hieman esimerkkejä niistä. Kolmannessa luvussa käydään läpi tutkimuksessa käytetyn IoT-yhdyskäytävän, Raspberry Pi:n käyttö IoT-sovelluksissa. Neljännessä luvussa käydään läpi tutkimuksessa käytetyn Googlen tarjoama sovelluskehitykseen tarkoitetun pilvipalvelun, Firebasen tärkeimmät ominaisuudet IoT:n kannalta. Esimerkit ja ohjeet pohjautuvat Firebasen tarjoamaan dokumentaation. Ominaisuuksia vertaillaan IoT-järjestelmältä odotettaviin ominaisuuksiin. Viidennessä luvussa käsitellään IoT-järjestelmän käyttöliittymän vaatimuksia sekä tiedon visualisointia. Kuudennessa luvussa esittelen käytännön tutkimukseni aiheesta sekä sen pohjalta tekemäni johtopäätökset.

# 2 MENETELMÄT

Tässä luvussa käyn lyhyesti läpi, mitä IoT käsitteenä tarkoittaa, mitkä ovat IoT-järjestelmän oleellisimmat tasot ja mitä teknologioita sekä käsitteitä IoT:hen liittyy. Tutkimuksen kannalta tärkeisiin aiheisiin perehdytään hieman tarkemmin.

## 2.1 IoT

*Esineiden Internet* eli IoT (engl. Internet of Things) kuvaa verkkoa, jossa Internet protokollan läpi älyä sisältävät, dataa keräävät tai muita toimintoja suorittavat laitteet kommunikoivat [Cirani *et al.* 2019, s.1].

Prosessoreiden ja muiden sähköisten komponenttien koon pienentyessä sekä niiden sähkönkulutuksen ja hintojen laskiessa jokapäiväisten laitteiden päivittäminen älylaitteiksi on yleistynyt [Floerkemeier ja Mattern 2010, s.1-2]. Langattomat rajapinnat mahdollistavat fyysisten laitteiden yksinkertaistamisen siirtämällä niiden käyttöliittymät verkkoon tai mobiilisovelluksiin [Floerkemeier ja Mattern 2010, s.5]. Kaupallisissa tarkoituksissa IoT muun muassa tehostaa liiketoimintamenettelyjä, vähentää logistiikkaan meneviä kuluja ja tarjoaa uusia liiketoimintamalleja. Kuluttajalle IoT tarjoaa elämää helpottavia, viihdyttäviä, ja turvaa parantavia laitteita. IoT:lla voidaan myös parantaa esimerkiksi liikenneturvaa [Floerkemeier ja Mattern 2010, s.6-7].

IoT:n suurimmat haasteet ohjelmistokehityksen kannalta ovat skaalautuvuudessa, yhteentoimivuuden luomisessa, käyttöönoton tekemisessä helpoksi, järjestelmien monimutkaisuudessa, tiedon määrässä ja tulkitsemisessa, turvallisuudessa, yksityisyydenturvassa sekä vikasietoisuuden kehittämisessä [Floerkemeier ja Mattern 2010, s. 7-8].

Nykyään IoT:ta sovelletaan jo useimmilla aloilla. Tärkeimmät alat ovat älykoti, älykaupunki, älykäs sähköverkko, liikenne, teollisuus, terveydenhuolto, älymaatalous, toimitusketjut ja vähittäiskauppa [Gour 2020].

## 2.2 IoT-Laitteet

Tyypillisiä ominaisuuksia, jotka erottavat IoT-laitteen tavallisesta laitteesta, ovat sen kykeneminen kommunikointiin Internetin ja usein myös toisten laitteiden kanssa, yleensä langattomilla teknologioilla. Laitteella on yleensä myös uniikki tunniste, jonka avulla se voi olla löydettävissä verkosta. IoT-laite pystyy joko jakamaan tietoa ympäristöstään tai vaikuttamaan siihen. IoT-laite voi myös tietää sijaintinsa, ja sen kanssa voi mahdollisesti kommunikoida käyttöliittymän läpi [Floerkemeier ja Mattern 2010, s.3].

IoT-laitteita ovat sensorit, aktuaattorit, yleiskäyttöiset tietokoneet, passiivisesti jäljitettävät esineet, sulautetut laitteet, IoT-yhdyskäytävät / hubit sekä mikrokontrollerit.

Sensorit havaitsevat muutoksia fyysisen ympäristönsä energiassa ja luovat dataa muuttamalla havaintonsa digitaalisiksi arvoiksi. Esimerkiksi liiketunnistin on turvajärjestelmissä käytetty sensori, joka havaitsee ihmisestä säteilevän infrapunan energian aiheuttamat muutokset. Lämpötilasensori taas käyttää lämpöenergiaa lämpötilalukeman luomiseen [Charlier *et al.* 2015, s.48-49].

Aktuaattorit ovat mekaanisia laitteita, kuten moottoreita tai kytkimiä, joita voidaan ohjata digitaalisesti. Esimerkiksi älylukko ja etäohjattava venttiili ovat aktuaattoreita [Charlier *et al.* 2015, s.52].

Yleiskäyttöiset tietokoneet kattavat tehokkaat, laskentaa varten kehitetyt PC:t. Nykyään myös laitteet, kuten älypuhelimet, tabletit ja pelikonsolit, voidaan laskea IoT-laitteiksi, sillä ne ovat yhdistettyinä Internetiin ja niitä voidaan käyttää IoT-palveluiden ohjaamiseen. Joskus ne soveltuvat myös käyttöön IoT-yhdyskäytävänä [Charlier *et al.* 2015, s.31].

Passiivisesti jäljitettävien esineiden kategoriaan kuuluvat esineet, jotka itsessään eivät ole kytkettyinä Internetiin, mutta jotka sisältävät Internetissä sijaitsevaan resurssiin yhdistettävän tunnisteen. Esimerkiksi RFID ja NFC-tagit luokitellaan tällaisiksi esineiksi. NFC-tagit (Near Field Communication) toimivat radiotaajuustunnistuksella, ja niiden avulla voidaan esimerkiksi luoda kaksisuuntainen kommunikaatio yhteensopivan älypuhelimen sekä itse tagin sisältävän laitteen kanssa [Charlier *et al.* 2015, s.42-45].

Sulautettuihin laitteisiin kuuluvat tiettyyn tehtävään erikoistuneet laitteet, kuten termostaatti tai vaaka. Tällaiset laitteet voivat sisältää mekaanisia osia, ja niitä käytetään usein korkeaa turvallisuutta vaativiin tehtäviin. Sulautetut laitteet yhdistetään Internetiin yleensä WiFi:n, älypuhelimen, yhdyskäytävän tai hubin avulla. Ne voivat myös sisältää myös fyysisiä, vuorovaikutuksen mahdollistavia toimintoja [Charlier *et al.* 2015, s.30-34].

IoT-yhdyskäytävät tai hubit toimivat IoT-järjestelmässä paikallisten, eli edge-laitteiden (sensorit ja aktuaattorit) ja internetin välikätenä. Useat IoT-laitteet käyttävät sovelluskohtaisia protokollia, lyhyen kantaman radiota (esimerkiksi ZigBee, Bluetooth tai WiFi -protokollat) tai pitkän kantaman radiota (LoRaWAN tai Sigford -protokollat) eivätkä ole yhdistettävissä Internetiin esimerkiksi ethernet-portin tai mobiilidatayhteyden avulla. Tällaisissa tilanteissa, kun laitteelta saatu data halutaan tallentaa pilveen tai jos laitetta tarvitse täytyy ohjata Internetin välityksellä, tarvitaan yhdyskäytävä tai hubi yhdistämään laite Internetiin [Adryan *et al.* 2017, s.132].

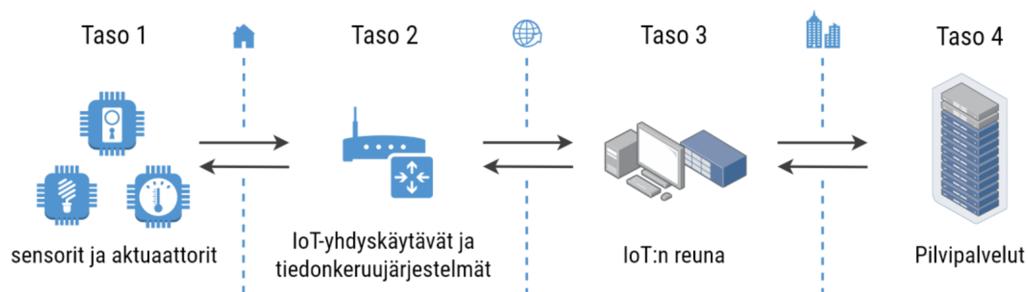
Yhdyskäytävän ominaisuudet voivat vaihdella käyttötarkoituksen mukaan: dataa voidaan prosessoida, suodattaa tai yhdistää, yhdyskäytävä voi mahdollistaa laitteiden välisen kommunikaation tai tuottaa diagnostiikkaa järjestelmästä [OAS 2020]. Yhdyskäytävillä on myös suuri merkitys laitteiden turvallisuuden kannalta. Yhdyskäytäviä käytetään niin kotiautomaatiossa kuin IIoT:ssa [i-SCOOP 2020a].

Mikrokontrollerit ovat virtapiirejä, jotka sisältävät mikroprosessorin, I/O-pinnit sekä muistia datalle ja ohjelmistolle. Niitä käytetään IoT:ssa muun muassa prosessoimaan päätelaitteilta saatua dataa. Mikrokontrollerit ovat hitaampia kuin yleiskäyttöiset tietokoneet, mutta halvempia, sekä kuluttavat vähemmän energiaa. Mikrokontrollereiden ohjelmointiin käytetään yleensä C tai C++ -ohjelmointikieltä. Mikrokontrollerin muistissa oleva ohjelma käynnistetään ns. bootloaderin toimesta automaattisesti laitteen käynnistyessä [Adryan *et al.* 2017, s.73].

## 2.3 IoT-Järjestelmän arkkitehtuuri

IoT-järjestelmän arkkitehtuuri muodostuu neljästä tasosta. Tasot esitetään kuvassa 1: ensimmäiset kaksi tasoa sijaitsevat lähiverkossa, ja viimeiset kaksi sijaitsevat palveluissa. Arkkitehtuurissa tieto voi kulkea tasojen läpi molempiin suuntiin, aina IoT:n reunalta asiakasohjelmaan, ja jokaisen tason välille vaaditaan sopivat menetelmät kommunikointiin.

Kuvan ensimmäinen taso kattaa ympäristöstään tietoa keräävät sensorit sekä ohjattavat aktuaattorit. Toinen taso sisältää laitteet, jotka keräävät sekä tarvittaessa muuntavat ensimmäisen tason laitteilta saadun datan ja ohjaavat sen Internetin läpi kolmannelle tasolle. Tällaisia laitteita ovat IoT-yhdyskäytävät sekä tiedonkeruujärjestelmät, jotka muuttavat analogista dataa digitaaliseksi. Toisen tason laitteet yleensä sijaitsevat fyysisesti ensimmäisen tason laitteiden lähettyvillä. Kolmannella tasolla tapahtuu yhdyskäytäviltä ja tiedonkeruujärjestelmiltä saadun datan esiprosessointi ja analytiikka IoT:n reunalla. Esiprosessointi IoT:n reunalla mahdollistaa sen hyödyllisen tiedon nopeamman erottelun, eikä ylimääräisen datan lähettämiselle datakeskukseen jää tarvetta. Neljännellä tasolla tapahtuu esiprosessoidun datan analysointi, prosessointi, hallinta sekä turvallinen säilytys datakeskuksessa tai pilvessä [Fuller 2020].




***Kuva 1.*** *Hahmotelma Fullerin esittämän arkkitehtuurin tasoista [Fuller 2020].*

## 2.4 IoT-protokollat

OSI-malli (engl. Open Systems Interconnection) on käsitteellinen malli, jonka tehtävänä on mahdollistaa järjestelmien välinen yhteentoimivuus standardisoimalla tiedonsiirtoon liittyviä toimintoja. OSI-pinoon kuuluu seitsemän tasoa. Kukin taso palvelee ylempää tasoa ja on samalla alemman tason palvelemana [Chou 2016, s.35].

OSI-pinon ensimmäiset kaksi kerrosta, fyysinen kerros (engl. physical layer) ja siirtokerros (engl. datalink layer) kattavat standardit, kuten ethernet ja WLAN.

Ethernet eli IEEE 802.3 on IEEE:n (engl. Institution of Electrical and Electronics Engineers) asettama standardi lähiverkon (LAN) teknologiasta. Se määrittelee, kuinka Ethernet-verkossa laitteiden välinen fyysinen, esimerkiksi sähkökaapelin läpi tapahtuva yhteys toteutetaan [Cirani *et al.* 2019, s.11-12].

WLAN (Wireless Local Area Network) eli IEEE 802.11 -standardi määrittelee lähiverkossa tapahtuvan radioaaltoja kommunikointiin käyttävien laitteiden välisen yhteyden. Wi-Fi-brändin standardit pohjautuvat WLAN-standardiin [Cirani *et al.* 2019, s.12-13]. Myös sandardit, kuten NFC, ZigBee, 4G ja LoRaWan, kuuluvat siirtokerrokselle [Chou 2016, s.37].

OSI-pinon kolmannelle kerrokselle, eli verkkokerrokselle (engl. network layer) kuuluu Internet protokolla.

Internet protokolla eli IP käsittää tietosähkeiden (engl. datagram) välittämisen Internetin läpi [Cirani *et al.* 2019, s.14-15]. Tietosähkeet ovat itsenäisiä viestejä, jotka sisältävät tiedon lähettäjästä ja vastaanottajasta [Techopedia 2020]. IP mahdollistaa yhdyskäytävien välisen kommunikaation ja on Internetin toiminnalle oleellisimpia protokollia. Tämänhetkinen versio, IPv4, välittää 32-bittisiä osoitteita, kun taas seuraavan generaation versio, IPv6, välittää 128-bittisiä osoitteita. Nämät IP-osoitteet asetetaan verkon asiakkaille DHCP-palvelimen toimesta [Cirani *et al.* 2019, s.14-15]. Verkkoprotokolla DHCP mahdollistaa IP-verkon laitteiden käyttää protokollia, kuten TCP ja UDP [Efficient IP 2020].

Neljännellä kerroksella, eli kuljetuskerroksella (engl. transport layer) on protokollat, kuten TCP ja UDP.

TCP ja UDP ovat kuljetuskerroksen yleisimmät protokollat. Ne tarjoavat sovelluksille palvelun asiakasohjelmien väliseen kommunikaatioon. TCP mahdollistaa luotettavan, yhteyspainotteisen tiedonsiirron, kun taas UDP on luotu nopeampaan ja yksinkertaisempaan viestien välittämiseen, jossa luotettavuuden merkitys on matalampi. Yksinkertaisuutensa vuoksi UDP on IoT-käytössä suosittu vaihtoehto. [Cirani *et al.* 2019, s.17-19]

Seitsemännes kerros, eli sovelluskerros (engl. application layer) kattaa protokollat, kuten HTTP, MQTT ja WebSocket.

HTTP (engl. Hypertext Transfel Protocol) on sovelluskerroksen käytetyin protokolla. HTTP toimii pyyntö–vastaus -periaateella: asiakasohjelma tekee palvelimelle pyynnön TCP-yhteyden läpi, ja palvelin vastaa pyyntöön. Pyynnössä lähetetty URL viittaa palvelimelta haluttavaan resurssiin. Pyynnössä käytettävä GET, POST, PUT tai DELETE -metodi kertoo palvelimelle käytettävän operaation tyypin. Pyyntö voi myös sisältää ylätunnisteita (engl. Header fields) antamaan lisätietoa pyynnön sisällöstä. Palvelin vastaa pyyntöön tilakoodilla, joka kertoo tuloksen pyynnön onnistumisesta. HTTP on ns. tilaton protokolla (engl. Stateless protocol) [Cirani *et al.* 2019, s.21-22]. Tilattoman protokollan mukaan pyynnön täytyy sisältää kaikki vaadittava informaatio halutun operaation suorittamiseen, koska palvelimella tai asiakasohjelmalla ei koskaan säilytetä pyyntöön liittyvää informaatiota [Cirani *et al.* 2019, s.86].

MQTT (engl. Message Queue Telemetry Transport) on julkaisu–tilaus (engl. publish/subscribe) -protokolla, joka käyttää TCP-protokollaa. MQTT-protokollassa julkaisijat lähettävät tietoa eräänlaiselle tiedon välittäjälle. Tilaajat taas voivat tehdä tilauksen tiedon välittäjälle, jolloin sinne saapuva tieto jaetaan välittömästi tilaajille. MQTT sopii IoT-tarkoitukseen hyvin, koska se on suunniteltu toimivaan tehokkaasti myös häviöllisessä verkossa. Tieto lähetetään binäärimuodossa, joten tiedon formaatti voi olla mikä vain [Adryan *et al.* 2017, s.337-339].

WebSocket-protokolla on kaksisuuntainen TCP-yhteys, joka mahdollistaa reaaliaikaisen kommunikaation palvelimen ja asiakasohjelman välille. WebSocketia käyttämällä voidaan välttää esimerkiksi HTTP-pohjaisista toteutuksissa käytettävät kiertokyselyt (engl. long polling) [Microsoft 2020a]. Kiertokysely on sovelluskehittäjien ratkaisu pyynnön aukipitämiselle, jotta palvelin voi lähettää päivittyvää tietoa asiakasohjelmalle, kun normaalisti HTTP mahdollistaa tiedon lähettämisen ainoastaan asiakasohjelman pyynnöstä [Hanson 2020]. WebSocket-yhteyden aukaisemiseen tarvitaan tietynlainen HTTP-pohjainen kättely asiakasohjelman ja palvelimen välillä. Kättelyn onnistuessa käytettävä protokolla päivittyy HTTP:sta WebSocketiksi, jonka jälkeen kaikki kommunikointi suoritetaan WebSocket-viestein aiemmin avatun samaisen TCP/IP-yhteyden avulla [Microsoft 2020a].

## 2.5 Yhteentoimivuus

IoT-järjestelmään kuuluu useita tasoja, joissa eri sovellukset tai laitteet vaihtavat tietoa keskenään. Ilman hyviä, sovelluksen kehityksen alkuvaiheilla valittuja käytäntöjä, pienetkin palvelimelle tehtävät muutokset voivat aiheuttaa sen, että tietoa ei voida enää käsitellä järjestelmän muilla tasoilla ilman tarvittavia muutoksia.

Yhteentoimivuus on käsite, joka kertoo vapaudesta ohjelmistojen välisessä tiedon välittämisessä. Jotta ohjelmistojen välinen yhteentoimivuus voidaan mahdollistaa, tarvitaan yhteiset käytännöt tiedon vaihtamiseen, tulkitsemiseen ja esittämiseen niin, että myös tiedon merkitys ei muutu [OmniSci 2020].

**REST**

REST (engl. Representational State Transfer) on web-kehityksessä käytetty arkkitehtuurinen tyylisuunta, joka määrittelee tietyt säännöt sovelluksen skaalautuvuuden ja varmatoimisuuden takaamiseksi. REST-arkkitehtuurissa oleellinen idea on, että asiakasohjelman täytyy pystyä tekemään pyyntö palvelimelle mahdollisimman vähällä tiedolla itse palvelimesta, jotta asiakasohjelman sovellus pysyisi riippumattomana palvelinpuolen toteutuksesta. Tämä mahdollistaa sen, että palvelinpuolella voidaan tehdä vapaasti muutoksia ilman, että asiaksohjelmaan tarvitsee tehdä muutoksia [Cirani *et al.* 2019, s.82-87].

REST-arkkitehtuurin sääntöihin kuuluu, että varsinaista resurssia ei koskaan siirretä palvelimen ja asiakasohjelman välillä, vaan ainoastaan senhetkinen esitys resurssista. Tätä kutsutaan resurssin representaatioksi. Representaatio voi olla esimerkiksi XML- tai JSON-muodossa. Toinen sääntö käsittää tilattoman protokollan, eli pyynnön täytyy sisältää tarvittava informaatio tiedon hakemiseen. Kolmas sääntö on, että haluttu tieto täytyy olla löydettävissä URI:n (engl. Uniform Resource Identifier) avulla. Neljännen säännön mukaan palvelimelta saadun tiedon representaation täytyy sisältää riittävät ohjeistukset asiakasohjelmalle, jotta tämä voi tarvittaessa sen perusteella toteuttaa seuraavan pyynnön palvelimelle. Sovellusta, joka toteuttaa REST-arkkitehtuurin säännöt, voidaan kutsua RESTful-sovelluksesi [Cirani *et al.* 2019, s.82-87].

**API**

API eli ohjelmointirajapinta (Application Programming Interface) on sovelluskehittäjille luotu rajapinta mahdollistamaan vuorovaikutukseen palvelun kanssa. Palvelun API:a käyttämällä sen rajapinnan tarjoamia ominaisuuksia voidaan hyödyntää omassa sovelluksessa. Ohjelmistoprojektin osia voidaan korvata käyttämällä ohjelmointirajapintoja, jolloin muihin alueisiin voidaan panostaa paremmin. API:en merkitys sovelluskehityksessä myös ohjelmistojen yhteentoimivuuden kannalta on suuri [Charlier *et al.* 2015, s.94].

**RESTful API**

RESTful API on palvelulle kehitetty ohjelmointirajapinta, joka noudattaa REST-arkkitehtuurin sääntöjä.

RESTful API ei vaadi minkään tietyn protokollan käyttämistä sovelluksessa. Yleisin RESTful ohjelmointirajapinnoissa käytetty protokolla on kuitenkin HTTP, ja

IoT-projekteissa sen käyttäminen on myös sen ominaisuuksiensa puolesta usein suositeltavaa [Adryan *et al.* 2017, s.324].

## 2.6 Web-teknologiat ja käsitteet

Perinteisiä web-teknologioita käytetään myös IoT-sovelluksissa, etenkin pilvipohjaisten järjestelmien käyttöliittymien kehittämisessä.

**HTML**

HTML (engl. Hypertext Markup Language) on merkintäkieli web-sivujen rakenteen ja sisällön esittämiseen. HTML-dokumentin rakenne koostuu elementeistä, joita esitetään aloitus- sekä lopetus-*tagilla*, eli tunnisteella. Elementit voivat muun muassa omistaa attribuutteja, sekä sisältää alielementtejä [Mozilla 2020a].

**JavaScript**

JavaScript on ohjelmointikieli, jota käytetään web-sivujen toiminnallisuuden kehittämiseen. Selaimessa JavaScriptiä voidaan käyttää esimerkiksi lisäämään web-sivulle interaktiivista sisältöä tai lähettämään pyyntöjä palvelimelle. Nykyään JavaScriptiä käytetään myös ohjelmointiin palvelinpuolella, esimerkiksi Node.js-ympäristössä [Javascript.info 2020].

Selaimessa on oma JavaScript-moottorinsa. Eri selaimet käyttävät omaa JavaScript-moottoriaan, esimerkiksi Chromen moottori on nimeltään V8 ja Firefoxin moottori on SpiderMonkey. Moottoreiden tuki tietyille ominaisuuksille voi vaihdella. JavaScriptille on myös oma määrittelynsä, ECMAScript [Javascript.info 2020]. Nykyinen ECMAScript-versio eli ES6 julkaistiin 2015 [W3Schools 2020].

**DOM**

Dokumenttioliomalli (engl. Document Object Model) eli DOM on ohjelmointirajapinta, jota käytetään HTML- ja XML-dokumenttien manipulointiin, esimerkiksi JavaScript-ohjelmointikielen avulla. Kaikki Web-sivun elementit esitetään DOM:issa olioina [Mozilla 2020b].

**Framework**

Ohjelmistokehykset (engl. frameworks) ovat valmiita, kehittäjille tarkoitettuja ohjelmistoja helpottamaan ohjelmistokehityksen prosessia. Ohjelmistokehyksen tarjoamia työkaluja käyttämällä kehittäjän ei tarvitse itse keskittyä sovelluksen matalan tason kehittämiseen. Toiminnonkulussa ohjelmistokehys kutsuu sovellukseen kirjoitettua koodia [Singh 2020].

## 2.7 Tietokannat

Oleellista sensoreilta tallennettavaa informaatiota ovat sensorin tunniste sekä perustiedot ja mittauksen aikaleima sekä arvo. Kun sensoreita on käytössä useampia ja dataa halutaan säästää pidemmältä aikaväliltä esimerkiksi analysointia varten, niin kerääntyneen tiedon määrä voi koitua ongelmaksi ilman tietokantaa. Myös tietyn tiedon hakeminen ilman valmiita työkaluja ja palvelinta voi olla hidasta [Adryan *et al.* 2017, s.349].

**Relaatiotietokannat**

Relaatiotietokannoissa tiedon hallinta tapahtuu relaatiotauluissa. Taulut voivat nimensä mukaisesti olla relaatioissa keskenään, ja niissä oleva tieto voidaan yhdistää toisen taulun tietoon avainten avulla. Esimerkiksi mittauksen lämpötila, aikaleima ja sensorin tyyppi voisivat sijaita taulussa nimeltä mittaus. Välineet-nimisessä taulussa sijaitsisi sensorin tyyppi ja perustiedot. Tällöin mittaus voitaisiin yhdistää välineeseen sensorin tyypin perusteella. Relaatiotietokantojen luominen, tiedon päivittäminen, ja sieltä tiedon hakeminen suoritetaan SQL-komennoilla (engl. Structured Query Language) [Adryan *et al.* 2017, s.349-351].

**Aikasarjatietokannat**

Aikasarjatietokantaan (engl. Time-Series Database) tieto tallennetaan aina tietyn hetken mukaan, ja hetkelle tallennetaan ainoastaan mittauksen nimi, arvo, laatu sekä aikaleima. Aikasarjatietokannan vahvuus on sen nopeus tiedon tallentamisessa sekä hakemisessa [Chou 2016, s.58].

**NoSQL-tietokannat**

NoSQL on tietokantamalli, jossa data tallennetaan muuten kuin SQL:sta tuttuihin relaatiotauluihin. NoSQL-tietokantoihin tieto tallennetaan yleensä JSON-objektien tyylisinä dokumentteina, graafeina tai avain-arvo -pareina, eikä tiedon tallentamisessa tarvitse noudattaa mitään ennalta määrättyä rakennetta. Tallennettava data voi koostua monikerroksisista tietorakenteista, joissa myös tiedon määrät voivat olla suuria. NoSQL:n hyödyt siis näkyvät kehityksen joustavuudessa ja skaalautuvuudessa tilan säästämisen sijasta [MongoDB 2020a].

MongoDB on esimerkki tietokannasta, joka käyttää NoSQL:ää. MongoDB:ssä data tallennetaan JSON-objektien tyylisesti dokumentteihin, jotka sijaitsevat kokoelmissa [MongoDB 2020b]. MongoDB:n voi asentaa omalle palvelimelle, tai vaihtoehtoisesti sitä voi käyttää MongoDB Atlas -pilvialustalta [MongoDB 2020c].

## 2.8 IoT-Alustat

IoT-alusta on IoT-järjestelmässä väliohjelmisto dataa luovien tai kuljettavien laitteiden (IoT-laitteet ja yhdyskäytävät) sekä sitä käyttävien sovellusten välillä [i-

SCOOP 2020b]. Kehittäjille IoT-alustat tarjoavat kehitystä nopeuttavia työkaluja, skaalattavuutta ja laiteriippumatonta yhteensopivuutta [KaaIoT 2020].

Pilvipalvelu myy käytännössä Internetin kautta käytettäviä palvelinkeskuksen resursseja, kuten tilaa datalle ja prosessoriaikaa, tietynlaisten valmiiden palvelintyökalujen ohessa. Työkaluja käytetään ohjelmistokehityspakettien tai ohjelmointirajapintojen kautta, ja niillä voidaan luoda esimerkiksi sovelluksen palvelinpuoli. IoT-sovelluksiin erikoistuneet pilvipalvelut voivat lisäksi tarjota työkaluja laitteen hallintaan, tiedon tallentamiseen ja analytiikkaan. IoT-alustoille ominaista on tietynlaiset viestinvälittäjät (engl. message broker), joilla data saadaan välitettyä tietokantaan tai reaaliaikaisesti esitettävään analytiikkaan [Adryan *et al.* 2017 s.134].

Suurimmat kulut datan laskennassa ja säilömisessä eivät synny tiedon hankkimisesta tai sen vaatimasta tilasta, vaan sen turvallisuuden, saatavuuden ja tehokkuuden hallinnasta sekä sen ympäristön vaihtamisesta. Turvallisen laskentaan ja säilöntään perustuvan infrastruktuurin ylläpitäminen maksaa moninkertaisesti sen ostamisen hinnan vuodessa. Pilvipalveluiden tarjoajilla on kuitenkin mahdollisuus keskittää työvaransa turvallisuuden, saatavuuden ja tehokkuuden prosessien automatisointiin palveluissaan, mikä mahdollistaa matalamman hinnan ja laadukkaammat palvelut [Chou 2016, s.62]. Suurimpiin pilvipalveluihin kuuluvat Amazon Cloud (AWS), Microsoft Cloud (Azure) ja Google Cloud [Adryan *et al.* 2017 s.134].

AWS IoT on Amazonin oma IoT-alusta, joka tarjoaa ratkaisuja teollisuuteen, kaupalliselle alalle ja älykotiin. Pilvipalveluiden lisäksi AWS IoT:n palveluihin sisältyy myös IoT-applikaatioihin tarkoitettu käyttöjärjestelmä mikrokontrollerille sekä edge-laitteille tarkoitettu ohjelmisto, joka tarjoaa työkalut aina prosessoinnista laitteiden väliseen kommunikaatioon paikallisesti [Amazon 2020].

Azure IoT on Microsoftin tarjoama IoT-alusta, joka tarjoaa kattavien pilvipalveluiden lisäksi palveluita IoT:n reunalle [Microsoft 2020b].

Google Cloud IoT on Googlen IoT-alusta, joka tarjoaa monipuoliset työkalut IoT:hen, kuten Cloud IoT Coren, Cloud ML Enginen koneoppimiseen ja BigQuery-tietovaraston [Google 2020å]. Cloud IoT Core mahdollistaa tiedon keräämisen laitteilta Googlen IoT-järjestelmään, jossa tietoa analysoidaan, visualisoidaan ja käytetään koneoppimiseen. Cloud IoT Core käyttää MQTT ja HTTP -protokollia [Google 2020a].

MongoDB Atlas on tietokanta palveluna (engl. Database as a Service, DBaaS) -tarjoaja, joka mahdollistaa perinteisten pilvipalveluiden ominaisuuksien lisäksi päivittyvän datan hakemisen tietokannasta reaaliaikaisesti sovelluksiin, ja

pilvifunktiot, joita voidaan asettaa käynnistymään haluttujen ehtojen täyttyessä. MongoDB Atlas käyttää MongoDB NoSQL-tietokantaa [MongoDB 2020d].

# 3 IOT-LAITEET

Tässä luvussa käsittelen lyhyesti IoT-laitteiden käyttöä IoT-projekteissa. Erityistarkastelussa on Raspberry Pi (lyh. RPI), sen IoT-soveltuvuuden sekä käyttötapauksien merkeissä.

## 3.1 Raspberry Pi

Raspberry Pi (Raspberry Pi Foundation) on yhden piirilevyn tietokone, jota käytetään esimerkiksi ohjelmoinnin opettamiseen ja kotiautomaatioon, mutta on myös suosittu teollisuuskäytössä. Raspberry Pi:n GPIO-pinneillä (engl. General Purpose Input/Output) voidaan ohjata sähkökomponentteja, mikä tekee siitä erinomaisen työkalun IoT-projekteissa [opensource.com 2020].

Esimerkiksi Raspberry Pi 3 Model B, joka on jo vanhempi malli, maksaa noin 45 euroa.

Raspberry Pi 3 Model B:ssä on kuuden ytimen 1.2GHz Broadcom BCM2837-prosessori, 1GB RAM, WLAN ja Bluetooth, Ethernet, 40-pinnin GPIO XXX, 4 x USB2-porttia, HDMI-portti ja Micro SD -portti käyttöjärjestelmää ja dataa varten [Raspberry Pi 2020a]. Raspberry Pi pohjautuu 32-bittisen ARM prosessoriin. ARM on RISC (Reduced Instruction Set) -tietokone, jossa on käytetty käskykannan luomiseen vähemmän transistoreita kuin normaaleissa prosessoreissa. Tällä saavutetaan halvempi hinta, matalammat lämmöt ja matala virrankäyttö [Chou 2016, s.22].

Raspberry Pi:n ns. virallinen käyttöjärjestelmä on Raspberry Pi OS (aiemmin Raspbian). *”Rasbian on ilmainen Debianiin pohjautuva käyttöjärjestelmä, joka on optimoitu Raspberry Pi:lle”* [Raspbian 2020]. Rasberry Pi OS on saatavilla 32-bittisenä Desktop eli työpöytäversiona ja Lite eli minimaalisena versiona [Raspberry Pi 2020b]. Muita RPI:n tukemia käyttöjärjestelmiä ovat muun muassa Fedora, Ubuntu MATE, Ubuntu Core, Windows 10 IoT Core, Debian, Arch Linux ARM ja Android Things [Cirani *et al.* 2019, s.321].

Raspberry Pi on erinomainen vaihtoehto proof-of-concept projekteihin sen yhdistettävyyden, käytössä olevien kehitystyökalujen sekä sen laajennettavuuden ansiosta, mutta sitä ei kuitenkaan aina haluta käyttää lopullisessa tuotteessa, kun halutaan säästää jättämällä esimerkiksi turhat osat ja ominaisuudet pois [Biron ja Follett 2016, s.32–34]. RPI:n sisältämä rauta sekä sille tarjolla olevat käyttöjärjestelmät mahdollistavat sen käyttämisen IoT-yhdyskäytävänä tai tiedon kerääjänä multiprosessointia apuna käyttäen [Cirani *et al.* 2019, s.321].

IoT-laitteista Raspberry Pi:n voi luokitella yleiskäyttöiseksi tietokoneeksi. RPI soveltuu hyvin käyttöön IoT-projekteissa, etenkin prototyyppisovelluksissa: se on

halpa, sisältää hyvät komponentit, ja sille on tarjolla useita soveltuvia käyttöjärjestelmiä sekä monipuolisia kehitystyökaluja.

## 3.2 Kehittäminen

Tässä kohdassa käyn läpi vaatimuksia yksinkertaistetusta käyttötavasta RPI:n käytöstä IoT-yhdyskäytävänä. Esimerkki perustuu oman tutkimukseni aiheeseen, ja siinä on pääosin pyritty noudattamaan IoT-yhdyskäytävälle ominaista asemaa ja tehtäviä IoT-järjestelmässä.

### Vaatimukset

Raspberry Pi:lle kehitetään ohjelmisto, jonka avulla se voi yhtäaikaisesti hakea tietoa useilta eri lähiverkon IoT-laitteilta, tässä tapauksessa Philips Hue-laitteiden sensoreilta, sekä myös ohjata lähiverkon laitteita, tässä tapauksessa Philips Hue-älyvaloja. Ohjelmiston tulee myös olla jatkokehitettävissä niin, että uusien laiteliityntöjen lisääminen ei vaikuta aiemmin toteutettuihin liityntöihin. RPI:n täytyy myös pystyä lähettämään tieto valittuun pilvipalveluun sekä vastaanottaa siltä tietoa. Pilvipalvelulta vastaanotettu tieto sisältää komentoja tai asetuksia laitteiden ohjaamiseen liittyen. Valitun ohjelmointikielen tulee olla Cloud Firestoren tukema.

Pilveen lähetettävän tiedon täytyy sisältää laitteen tunniste, nimi, tyyppi, laitteen tavoitettavuus, mittauksen aikaleima ja itse mittaus. Tietoa tallennettaessa käytetään myös itse yhdyskäytävän tunnusta. Laitetta ohjataan laitteen tyypin ja tunnuksen perusteella.

Laitteen täytyy myös olla helposti käyttöönotettavissa konfiguraatioiden avulla, ja sen käyttäminen käyttönoton jälkeen ei saa vaatia ylimääräistä vuorovaikutusta käyttäjältä, esimerkiksi käynnistyksen yhteydessä. Sovelluksen täytyy pystyä toimimaan yhtäjatkoisesti taustaprosessina, eikä se saa aiheuttaa liiallista ylikuumenemista RPI:lle.

### Kehittäminen

Tämä alakohta esittelee pääpiirteet yhdelle tavalle täyttää aiemmassa alakohdassa esitetyt vaatimukset. Teknologioiden valinnassa on pyritty käyttämään helposti lähestyttäviä, hyvän tuen tarjoavia teknologioita, jotka soveltuvat prototyypin kehittämiseen. IoT-sovelluksiin paremmin erikoistuneita käyttöjärjestelmiä sekä ohjelmointityökalujakin on: esimerkiksi Node-RED on Node.js:n päälle rakennettu ohjelmointityökalu, joka soveltuu erityisen hyvin IoT-tarkoituksiin. Node-RED:istä saa lisätietoa osoitteesta: https://nodered.org/.

Vaiheet RPI:n käyttämiseen IoT-yhdyskäytävänä ovat sopivan käyttöjärjestelmän (ja kehitysympäristön) asentaminen, laitteiden välisen

kommunikaation luominen, lähdekoodin luominen sensoreiden ja laitteiden ohjaamiseen sekä pilveen kirjoittamista varten [Poyen 2019].

Jotta RPI:tä voidaan käyttää yhdyskäytävänä, sille täytyy asentaa asianmukainen käyttöjärjestelmä. Tarpeesta riippuen suurin osa aiemmin esitetyistä käyttöjärjestelmistä on sovellettavissa IoT-tarkoituksiin. Raspberry Pi OS on kuitenkin sen virallisen asemansa johdosta hyvä vaihtoehto; Raspberry Pi OS on hyvin optimoitu, sekä sille on hyvä tuki. Raspberry Pi OS:lle on myös tarjolla kattavat ohjelmistotyökalut. Koska graafiselle käyttöliittymälle ei ole tapauksessa tarvetta, niin Raspberry Pi OS:n Lite -versio sopii tarkoitukseen hyvin, myös keveytensä vuoksi. Raspberry Pi OS:n levykuva voidaan asentaa osoitteesta: https://www.raspberrypi.org/downloads/raspberry-pi-os/.

Käyttöjärjestelmä tulee asentaa Micro SD -kortille, joka toimii myös laitteen tallennustilana. Levykuvan asentamisessa Micro SD -kortille voidaan käyttää Windowsilla balenaEtcher-sovellusta: https://www.raspberrypi.org/documentation/installation/installing-images/windows.md.

Jos RPI:tä käytetään komentorivin kautta, niin RPI:lle kannattaa aktivoida SSH-protokolla. Se mahdollistaa RPI:n etäohjaamisen komentorivin avulla, esimerkiksi lähiverkossa sijaitsevalta Windows-tietokoneelta. SSH-yhteyden luomiseen voidaan käyttää esimerkiksi PuTTY:a. PuTTY voidaan asentaa seuraavasta osoitteesta: https://www.putty.org/

SSH voidaan aktivoida RPI:n asetuksista. RPI:n asetuksiin Raspberry Pi OS:ssa pääsee käsiksi seuraavalla komennolla:

```
sudo raspi-config
```

Samalla asetuksista voi aktivoida käytettävät fyysiset rajapinnat, kuten GPIO:n. SFTP:n (engl. Secure File Transfer Protocol) käyttäminen on suositeltavaa tiedostojen siirtämisessä RPI:lle, erityisesti kun kehittäminen halutaan tehdä erillisellä tietokoneella. WinSCP on yksi vaihtoehto SFTP-ohjelmalle Windowsissa: https://winscp.net

Jos kehityksessä käytetään Python-ohjelmointikieltä, niin projektille voidaan luoda oma virtuaaliympäristönsä helpottamaan pakettien hallintaa. Esimerkiksi Python3-kielelle virtuaaliympäristön luomisen ja käyttämisen komennot ovat [PyPA 2020]:

**Virtuaaliympäristön luominen:** `python3 -m venv <projektikansio>`
**Virtuaaliympäristön aktivointi:** `source <projektikansio>/bin/activate`
**Paketeiden listauksen luominen:** `pip freeze > requirements.txt`

Vaatimusten täyttämisessä ohjelmiston kannalta keskeisiä teemoja ovat: kuinka yhdyskäytävä konfiguroidaan, kuinka laitteiden kanssa kommunikoidaan ja kuinka ne haetaan ohjelmistoon sekä kuinka jatkuva tiedon hakeminen ja laitteiden ohjaaminen yhtäaikaisesti toteutetaan.

Konfiguraatiot luetaan ohjelmiston käynnistyessä. Konfiguraatiot kannattaa kirjoittaa ihmisen luettavaan formaattiin, joka voidaan helposti lukea myös ohjelmaan. Yhteentoimivuuden sekä sovelluksen kehitettävyyden kannalta on hyvä käyttää formaatteja, joita käytetään muissakin ohjelmiston osissa. Esimerkiksi JSON täyttää tässä tapauksessa nämät piirteet. Konfiguraatiotiedostoon voidaan täyttää esimerkiksi pilvipalvelun vaatimat informaatiot, kuten laitteen nimi.

Laitekommunikaatiolle voi luoda oman Python-moduulinsa. Moduuli sisältää kaikki vaadittavat metodit tiedon hakemiseen laitteelta, sekä myös metodit tiedon päivittämiseen laitteelle. Tietoa käytetään sovelluksen alustusvaiheessa alustamaan pilvipalvelun tietokantaan tiedot käytettävissä olevista laitteista, sekä aloittamaan Python sovelluksessa tehtävät kunkin laitteen tiedon hakua ja ohjaamista varten. Jos mahdollista, sensoridataa tallennettaessa kannattaa käyttää sensorilta saatua aikaleimaa mittausten ajankohtien todenmukaisuuden säilyttämiseksi.

Samanaikaisten tehtävien luomiseen voidaan käyttää Pythonin threading-moduulia. Threading-kirjaston avulla sovellukseen voidaan luoda useita säikeitä. Säikeet mahdollistavat tehtävien ajamisen sovelluksessa yhtäaikaisesti. Säikeet soveltuvat käyttöön, jos tehtävään ei sisälly prosessorin kannalta raskaita tehtäviä, vaan esimerkiksi ajoitettuja tehtäviä. CPU-raskaisiin tehtäviin kannattaa käyttää moniprosessointia. Moniprosessointi voidaan toteuttaa Pythonin multiprocessing-moduulilla [Anderson 2020].

Sovelluksen automaattiseen käynnistykseen voidaan käyttää *crontab*-taulukkoa. Taulukon avulla prosesseja voidaan ajastaa käynnistymään haluttuna ajankohtana. Esimerkiksi seuraava rivi käynnistää shell-skriptin RPI:n käynnistymisen yhteydessä [Dexter Industries 2020]:

```
@reboot /home/pi/start_project.sh
```

Shell-skriptissä täytyy ensin aktivoida projektin virtuaaliympäristö sekä käynnistää itse ohjelma taustaprosessiksi esimerkiksi *screen*-komennolla:

```
screen -d -m python3 app.py
```

Pilvipalvelun yhteyksiä varten tulee käyttää Firestore SDK:n tarjoamia metodeja. Firestore SDK:n käyttämiseen perehdytään tarkemmin luvussa 4. Seuraavassa kohdassa käsitellään Philips Hue-älyvaloja, sekä muita Hue laitteita.

Kommunikaatio Hue laitteiden kanssa sovelluksessa toteutetaan niille kehitettyä RESTful API:a apuna käyttäen.

## 3.3 Philips Hue ja RESTful API

Philips Hue on Philipsin kehittämä sarja älyvalolaitteita. Huen perinteisimmät älyvalot toimivat joko Bluetoothin tai Hue-sillan (Hue Bridge) avulla. Bluetoothia käyttämällä valoja voidaan kontrolloida suoraan älypuhelimelle asennettavasta Hue sovelluksesta. Hue siltaa käyttämällä valot voidaan kytkeä sillan ZigBee-verkkoon, jolloin ohjaaminen tapahtuu sillan läpi [Philips 2020a].

Silta mahdollistaa valoille ns. *solmuverkon* luomisen *Zigbee Light Link* -protokollan avulla. Solmuverkossa laitteet voivat kommunikoida keskenään ja välittää toistensa kautta viestejä sillalle. Sillan avulla laitteet voidaan myös yhdistää Internetiin, jolloin valoja voidaan ohjata Hue Portaalin avulla. Philips Hue-sillassa on myös sisäänrakennettu RESTful API, joka mahdollistaa Hue-valojen ja -laitteiden ohjaamisen omasta sovelluksesta, kunhan sovellus sijaitsee samassa lähiverkossa sillan kanssa [Philips 2020b].

Jotta Huen RESTful API voidaan ottaa käyttöön sovelluksessa, tarvitaan Hue-sillan lähiverkon IP-osoite sekä API-käyttäjänimi. Käyttäjänimi voidaan generoida esimerkiksi Huen *CLIP API Debuggerin* eli ”virheidenjäljittäjän” avulla. Debuggerin saa auki lähiverkon osoitteesta: `https://<sillan-ip>/debug/clip.html`. Huen RESTful API käyttää HTTP:tä, joten pyynnöissä käytetään HTTP-metodeja [Philips 2020d]:

**GET** hakee resurssin tiedot
**PUT** muuttaa resurssin tietoa
**POST** lisää uuden resurssin
**DELETE** poistaa resurssin

Resurssin osoite alkaa aina sillan IP:llä ja api-osalla, sekä yleensä myös tarvitaan käyttäjänimi: http://<sillan-ip>/api/<käyttäjänimi>. Käyttäjänimen jälkeen osoitteeseen lisätään haluttu resurssi, kuten lights, config tai sensors. Kun käytetään esimerkiksi POST tai PUT -metodia, niin resurssin tiedot siirretään viestin ”Body”-osassa JSON-muodossa [Philips 2020c].

Esimerkiksi Hue-valon tilaa ja kirkkautta muutettaessa viestin Body-osa voisi näyttää seuraavalta:

```
{ "on" : true, "bri” : 150 }
```

on-avaimelle asetettu arvo kertoo valon kytkennän tilan ja bri-avaimelle asetettu arvo kirkkauden. Haluttu laitteen ominaisuus tai tieto pyynnön onnistumisesta palautetaan palvelimen vastauksessa. Jos Hue-valoja halutaan ohjata Python-sovelluksesta, niin apuna kannattaa käyttää requests ja json-kirjastoja.

Requests-kirjasto on tehty helpottamaan HTTP-pyyntöjen lähettämistä, muun muassa poistamalla tarpeen itse kirjoittaa tai muotoilla pyynnön merkkijonoa [Requests 2020]. Requests-kirjasto täytyy ladata erikseen Pythonin pakettien asentajalla, pip:llä. Json-kirjastoa taas voidaan käyttää koodaamaan ja dekoodaamaan JSON-objekteja [Python 2020].

Python sovelluksessa aiempi esimerkki toteutettaisiin PUT-metodilla seuraavasti:

```
body = {
    "on": true,
    "bri": 150
}

request = requests.put("http://sillan-ip
    /api/username/lights/1/state", data = json.dumps(body))

print(request.json())
```

Sillalta kutsuun saatu vastaus kirjoitetaan request-muuttujaan.

Hue-älyvalot löytyvät RESTful API:sta /lights-osoitteen alta. Valon ominaisuuksiin kuuluu muun muassa valon kirkkaus sekä valon päällekytkentä. Kirkkautta voidaan säätää asettamalla API:sta valon bri-avaimelle arvo väliltä 0-254. Päällekytkentä asetetaan avaimelle on totuusarvona. Erityyppiset valot voidat sisältää muita parametreja, esimerkiksi valon sävyn säätämiseen. Valon tyyppi kirjoitetaan avaimelle type. Tavallinen valo löytyy API:sta tyypillä ”Dimmable light”.

Myös Hue älypistorasia (Hue smart plug) löytyy valojen osoitteen alta. Älypistorasiasta tarvitsee käyttää lähinnä päällekytkentää, joka löytyy pistorasian avaimelta on. Älypistorasian tyyppi on ”On/Off plug-in unit”.

Philips Hue -liiketunnistin sisältää itse liiketunnistimen lisäksi valoisuusanturin sekä lämpötilasensorin. Kullakin sensorilla on oma avaimensa API:sta /sensors-osoitteen alla. Jokaisella sensorille löytyy avaimet tyypille, mittausajankohdalle ja tunnisteelle lukuisten muiden ominaisuuksien lisäksi.

Itse liiketunnistimen tärkein tieto on *presence* eli läsnäolo, joka kertoo havaitusta liikkeestä. Läsnäolo tallennetaan totuusarvona, ja siitä tallennetaan myös aikaleima *lastupdated*-avaimelle. Tieto päivitetään API:in ainoastaan kun totuusarvo muuttuu eli kun liikettä havaitaan tai kun liike loppuu. Liikkeen

loppumisen jälkeen sensori odottaa noin kymmenen sekuntia ennen kuin läsnäolo päivitetään epätodeksi. Sensorin config-avaimen alta voidaan myös muuttaa sensorin havainnoinnin herkkyyttä. Liiketunnistin on helppo tunnistaa API:sta käyttämällä sen avaimen tyyppiä ”ZLLPresence”.

Lämpötilasensorin mittaus tallennetaan avaimelle *temperature*. Mittaus esitetään Celsiusasteina kahden desimaalin tarkkuudella kokonaislukuna (esimerkiksi 22,17°C = 2217). Lämpötilasensorin tyyppi on ”ZLLTemperature”.

Valoisuusanturin mittaus kirjoitetaan avaimelle *lightlevel*. Anturilla on myös avaimet *dark* ja *daylight*, jotka esitetään totuusarvoina. Valoisuusanturin asetuksista voi asettaa kynnysarvon avaimelle *tholddark*. Kun valoisuusaste ylittää kynnysarvon, *daylight* asetetaan todeksi ja *dark* epätodeksi. Valoisuusanturia voidaan käyttää käynnistämään Hue valot näiden parametrien mukaan. Anturin tyyppi API:ssa on ”ZLLLightLevel”.

## 3.4 Yhteenveto

Tässä kappaleessa käytiin läpi ratkaisuja Raspberry Pi:n käyttämiseen IoT-yhdyskäytävänä sekä kommunikointia Hue-laitteiden kanssa RESTful API:n avulla. IoT-laitteilta saatu tieto täytyy siirtää lähiverkosta eteenpäin, mahdollisesti prosessoida ja säilöä seuraavalla tasolla. Tiedon säilöminen, ja sieltä eteenpäin siirtäminen voidaan tehdä IoT-alustalla pilvessä. IoT-alustaan tieto siirretään pilvialustan tukemilla protokollilla, ja toteutetaan sen tarjoamien kehitystyökalujen avulla.

# 4 PILVIALUSTA - FIREBASE

Tässä luvussa käsittelen Firebase-pilvialustan sisältöä, esitän ohjeita sen käyttöönottamiseen sekä käyn läpi sen mahdollisuuksia sovelluskehityksessä. Firebasen esittely on tutkielman kannalta oleellista, sillä käytin sitä projektissani IoT-alustana tiedon säilömiseen ja kommunikaatioon laitteiden sekä käyttöliittymän välillä. Tässä luvussa perehdytään tarkemmin Firebasen sovelluskehitysalustaan palvelinkäytössä, ja luvussa 5. asiakasohjelman kehittämisessä.

Firebase on Googlen omistama sovelluskehittäjille suunnattu pilvialusta helpottamaan sovellusten kehitysprosessia poistamalla tarpeen oman infrastruktuurin luomiselle sekä ylläpidolle. Firebase on alun perin Andrew Leen ja James Tamplin (Firebase Inc.) vuonna 2011 perustama alusta, mutta Google hankki sen yrityskaupassa vuonna 2014. Firebasesta on sittemmin tullut Googlen ”lippulaiva” web- ja mobiilisovelluskehittämiseen [Crunchbase 2020].

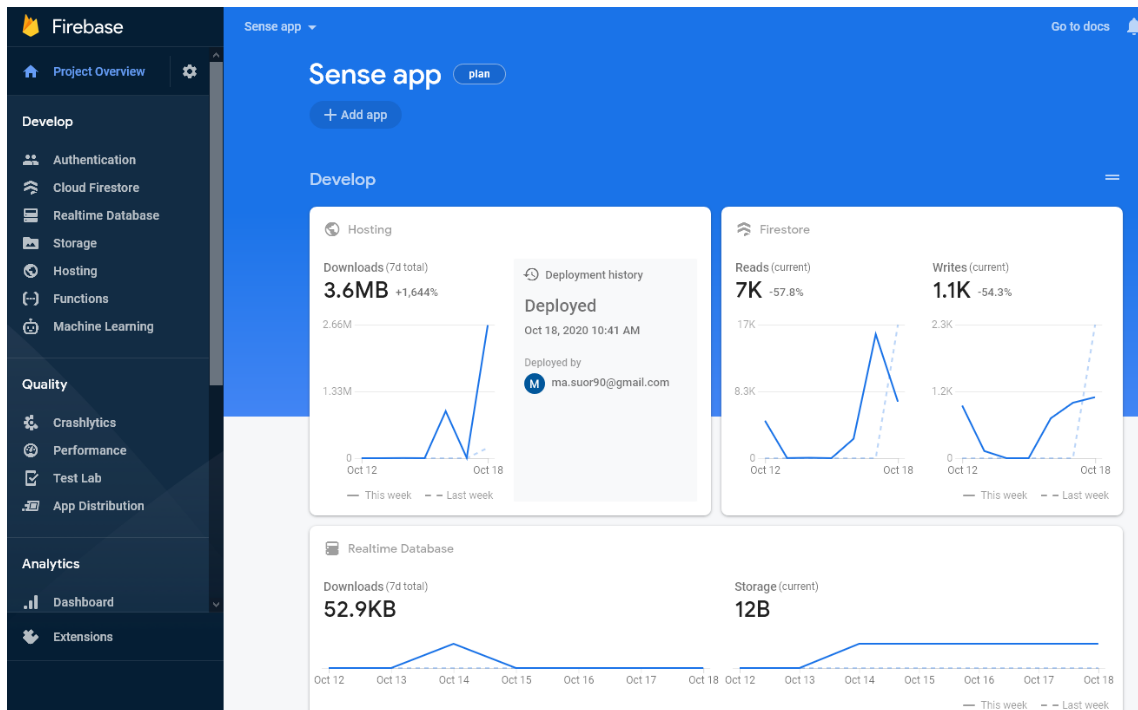
Firebasen tärkeimpiä ominaisuuksia ovat nykyään sen tietokannat ja niiden käyttöön tarkoitettu sovelluskehityspaketti. Tietokantojen lisäksi Firebase mahdollistaa pilvessä sovellukselle käyttäjän todentamisen, pilvifunktiot, tiedostojen säilyttämisen, koneoppimisen, hostingin, analytiikat ja sovelluksen testaamisen sekä suorituskyvyn seurannan [Google 2020b].

Firebasen tarjoamat työkalut koneoppimiseen ovat käytössä mobiili SDK:n kautta. Tarjolla on valmiit API:t konenäköön, kuten tekstin tulkitseminen ja kuvien luokittelu, kielen tulkintaan ja kääntämiseen sekä myös omien mallien käyttäminen on mahdollista. Firebasella on omat koneoppimisen API:nsa sekä pilvikäyttöön että mobiililaitekäyttöön [Google 2020c]. Koska Firebasen Koneoppimistyökalujen käyttö vahvasti painottuu mobiililaitteille ja koska koneoppiminen ei ole tutkimuksessani tärkeässä roolissa, niin Firestoren tarjoamat koneoppimisen ominaisuudet jätetään käsittelemättä tarkemmin.

## 4.1 Firebase console

Omaa Firebase-projektin luontia varten tarvitaan Google-tili. Kirjautumisen jälkeen projekti voidaan luoda kuvassa 2 esitetystä Firebase konsolista: https://console.firebase.google.com/. Konsolista hoidetaan myös kaikki muu projektiin liittyvä hallinta ja asetukset. Projektin luonnin yhteydessä projektille määritellään uniikki tunniste, joka myös toimii osana domain-verkkotunnusta, jos sovellus halutaan julkaista Firebasen tarjoamassa hosting-palvelussa. Projektin luonnin yhteydessä voidaan myös valita, halutaanko projektissa käyttää analytiikkaa. Projektin luominen ei vaadi pankkitunnuksia, vaan aktivoi ilmaisen ”Spark”-maksusuunnitelman, joka mahdollistaa esimerkkinä Cloud Firestore -

tietokantaan kirjoituksia 20000/pv, lukuja 50000/pv ja poistoja 20000/pv. Ilmaiseen suunnitelmaan kuuluu myös 10GB tallennustilaa ja palvelusta lataaminen on rajoitettu 10GB/kk. Rajoituksiin lasketaan kaikkien käyttäjien toiminta sovelluksessa. Rajojen täyttyessä palvelun toiminta rajoittuu hetkelliseksi eikä liikakäyttö ole mahdollista. Maksullisessa Blaze-sopimuksessa Firestoren hinta määräytyy käytön mukaan: esimerkkinä palvelusta laskutetaan $0.18/100k kirjoitusta, $0.06/100k lukua ja $0.02/100k poistoa. Myös pilvifunktioiden käytöstä laskutetaan tietyn käyttömäärän jälkeen [Google 2020d].




***Kuva 2.*** *Projektin hallinta tapahtuu Firebase konsolista [Google 2020ä].*

Firebase konsolin etusivulta näkee sovelluksen käyttöön liittyvää informaatiota sekä linkit sen tärkeimpiin toimintoihin. Kehitystä varten konsolista tulee valita, halutaanko käyttää reaaliaikatietokantaa vai Cloud Firestorea. Myös molempien yhtäaikainen käyttö projektissa on mahdollista. Cloud Firestore -osiosta voidaan tarkastella omaa Firestore tietokantaa, sen sääntöjä, tietokantaindeksejä sekä käyttöä. Realtime Databasen alta taas voidaan katsella reaaliaikatietokannan dataa, sääntöjä, varmuuskopioita ja käyttöä. Asetuksien alta löytää muun muassa Web API -avaimen, jota tarvitaan sovelluksen yhdistämiseksi Firebase-projektiin, integraatioiden luomisen muihin palveluihin ja Admin SDK -avaimen luomisen. Authentication-osion alta voi luoda sovellukselle käyttäjät, muuttaa käyttäjiin liittyviä asetuksia sekä hakea käyttäjäkohtaisen UID:n, jota voidaan käyttää käyttäjän tunnistamiseen sovelluksessa, kuten esimerkiksi käyttäjäoikeuksien tai tiedoston omistajan tarkastelussa. Storage-osiosta voidaan hallita palveluun

tallennettuja tiedostoja, niille luotuja sääntöjä sekä seurata niiden käyttöä. Jos Firebasen hosting-palvelua halutaan käyttää projektissa, niin hosting-osio tarjoaa julkaistujen versioiden historian seuraamisen sekä tarvittaessa niiden käyttöön palautukset. Functions-osion alla on projektissa käytettävien pilvifunktioiden informaatiota ja Machine Learning -välilehden alla tietoa Firebasen koneoppimiseen tarkoitetuista API:sta ja projektissa jo käytössä olevista koneoppimismalleista. Quality-osion alla on sovelluksen laadun varmistamiseen liittyviä ominaisuuksia, kuten tehokkuuden- ja sovelluksen testaaminen sekä kaatumisilmoitukset. Analytics-osion alla on Googlen analytiikkaan liittyviä ominaisuuksia ja näkymiä.

Firebasen valmiit analytiikka-ominaisuudet ovat pääasiassa tarkoitettu sovelluksen käyttöön sekä käyttäjän toimintaan liittyvään seuraamiseen [Google 2020e].

## 4.2 Tietokannat

Firebasen tarjoamat tietokannat ovat oleellisessa roolissa tutkimuksessani. Koska myös Cloud Firestore -tietokanta tukee reaaliaikaista tiedon synkronointia, niin tästä lähtien viittaan Firebasen reaaliaikatietokantaan (engl. Realtime Database) lyhenteellä RTDB, jotta säästytään mahdollisilta sekaannuksilta. Reaaliaikaisuuden lisäksi yhteistä Firestorella ja RTDB:llä on niiden käyttämä NoSQL-tietokantamalli. Molemmille on tarjolla myös omat SDK:nsa.

RTDB sopii pienempiin projekteihin, joissa tiedon määrä on pientä ja tietorakenteet yksinkertaisempia. Firestoren käyttö taas mahdollistaa edistyneemmät työkalut, esimerkiksi kyselyihin ja lajitteluun, monimutkaisemmat tietorakenteet sekä paremman saatavuuden. Toisaalta RTDB mahdollistaa asiakasohjelman läsnäolon eli yhteyden tilan seuraamisen, sekä sen käytön hinnoittelussa ei oteta huomioon tietokantaan tehtäviä luku, kirjoitus tai poisto-operaatioita, vaan ainoastaan tiedonsiirto sekä tallennustilan käyttö [Google 2020f].

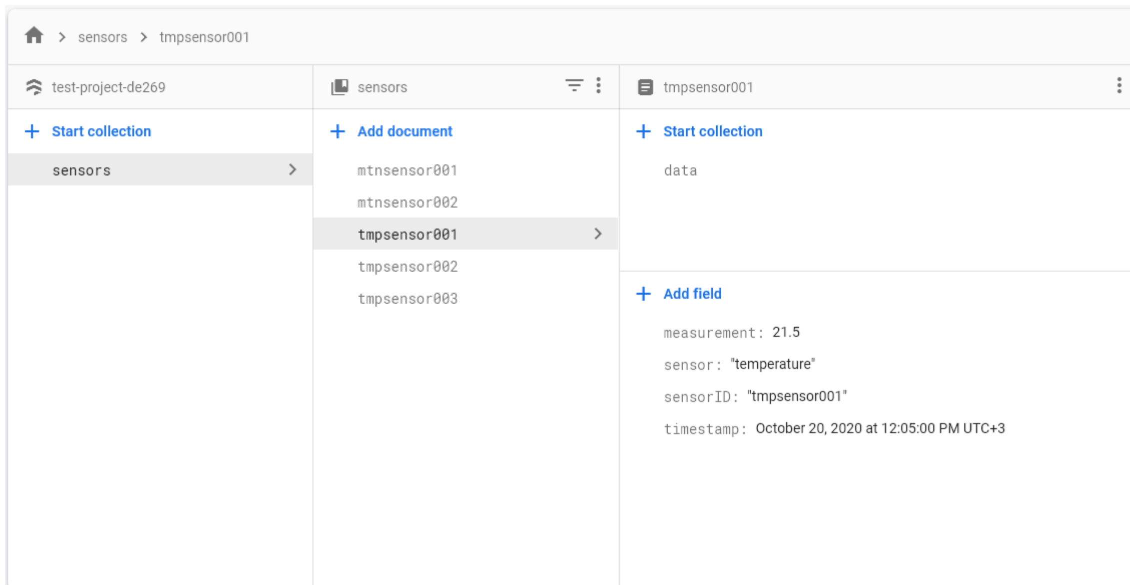
Koska RTDB:n käyttöä ei laskuteta tietokantatransaktioiden mukaan, niin se sopii hyvin pieniin IoT-projekteihin, joissa esimerkiksi tarvitaan vain tuoreinta – reaaliaikaista dataa, vaikkapa infonäytöllä esittämiseen. Koska projektissani kuitenkin käytin Cloud Firestore -tietokantaa, niin tulevat esimerkit ja ohjeet jätetään pääasiassa käsittelemättä RTDB:n osalta.

### 4.2.1 Cloud Firestore

Cloud Firestore on Firebasen pilvessä sijaitseva NoSQL-tietokanta. Firestore käyttää NoSQL-tietokantamallia, jossa data tallennetaan kokoelmissa sijaitseviin dokumentteihin. Kullakin dokumentilla voi olla useista tietotyypeistä valittavia

avain-arvo -pareja, ja näiden lisäksi dokumentit voivat myös sisältää alikokoelmia omine dokumentteineen [Google 2020g].

Esimerkiksi IoT-sovelluksen tietokannassa voi olla kokoelma nimeltä *sensors*. Tässä kokoelmassa kukin dokumentti kuvaisi yhtä sensoria, jolla voisi olla kenttä – arvo -pareina esimerkiksi *sensor : "temperature"* ja *measurement : 21.5*. Sensor-dokumentti voisi myös sisältää kokoelman nimeltä data, johon kirjattaisiin kaikki aiemmat mittaukset [Google 2020h]. Kuvassa 3 on esimerkki Firestoreen tehdystä tietokannasta Firebase-konsolista katseltuna.




***Kuva 3.*** *Esimerkki tietokannan rakenteesta [Google 2020ä].*

Vaikka Firestoressa sijaitsevat dokumentit voivatkin sisältää monimutkaisia tietorakenteita, niin sieltä tiedon hakeminen tietokantakyselyillä on tehokasta: tietoa voidaan noutaa Firestoresta myös dokumenttitasolla niin, että kokoelman muut dokumentit ja alikokoelmat jätetään täysin noutamatta. Data voidaan noutaa myös reaaliaikaisesti käyttämällä tietynlaisia kuuntelijoita, jotka ilmoittavat uusista muutoksista tietokannassa. Dokumentin, jolle on asetettu kuuntelija, muutokset saadaan automaattisesti "tilannekuvassa" (engl. Snapshot) sovellukseen, jolloin tietoa ei tarvitse hakea erikseen. Haettavien tuloksien rajoittamiseen ja suodattamiseen on myös omat työkalunsa [Google 2020i].

Firestorea luotaessa tietokannalle täytyy valita sijainti Firebasen konsolin kautta. Oman Cloud Firestore -tietokannan sijainti kannattaa valita käyttäjäkunnan mukaan. Sijainti on mahdollista valita Pohjois-Amerikasta, Etelä-Amerikasta, Euroopasta, Aasiasta tai Australiasta. Euroopassa palvelimet ovat Lontoossa, Frankfurtissa ja Zürichissä. Valitsemalla sopivan sijainnin voidaan vähentää latenssia ja parantaa saavutettavuutta [Google 2020j].

### 4.2.2 Ohjelmistokehityspaketit

Firestorea voidaan käyttää sovelluksissa sitä varten tehdyn SDK:n avulla. SDK on tarjolla ainakin seuraaviin ohjelmointikieliin tai ympäristöihin: Node.JS, Java, Python, Unity, C++, Ruby, PHP ja Go [Google 2020i].

Cloud Firestore -tietokantaa luotaessa täytyy valita tuotantotilan tai testitilan väliltä: tuontantotilassa kaikki asiakasohjelmien kyselyt ja kirjoitukset tietokantaan estetään oletuksena. Testitilassa kaikki luvut ja kirjoitukset sallitaan 30:n päivän ajan, jonka jälkeen ne kielletään ellei turvasääntöjä ole muutettu. Turvasääntöihin palataan tässä luvussa myöhemmin.

Kehityksen alussa täytyy valita sopiva alusta käyttötarkoituksen mukaan. Jos esimerkiksi kehitetään ohjelmistoa, joka vaatii raskaampaa laskentaa ja monipuolisia palvelinpuolen ominaisuuksia, niin Node.js voi olla hyvä valinta sovelluksen palvelinpuolelle. Jos halutaan kehittää mobiilisovellus Android-käyttöjärjestelmälle, niin vaihtoehtoina ovat esimerkiksi Java tai Kotlin+KTX. Jos kehitetään web-sovellusta, joka ei vaadi monimutkaista backend-koodia, niin pelkän Web-SDK:n käyttäminen voi riittää. Samaa tietokantaa käyttämään voidaan myös luoda sovellukset useammalle alustalle.

On hyvä ottaa huomioon erot ohjelmointikielten ohjelmistokehityspaketeissa. Myös jotkin perusominaisuudet eivät ole tuettuina kaikilla ohjelmointikielillä. Tämän lisäksi Admin SDK on tarkoitettu lähinnä palvelinkäyttöön, jolloin tuettuja kieliä ovat vain Node.js, Java, Python, Go ja C#. Admin SDK:n käyttö mahdollistaa palvelinpuolelle tyypillisiä ominaisuuksia, kuten käyttäjien hallinnan [Google 2020k].

### 4.2.3 Kehittäminen

Tässä alakohdassa käyn läpi Firestoren käyttöönottamisen ohjelmointiympäristössä sekä esittelen oleellisimmat Firestoren käyttöön liittyvät operaatiot. Esimerkit esitetään Python-ohjelmointikielellä, sillä se oli myös tutkimuksessani tärkeässä osassa kehitystä.

Python kehittämiseen tarvitaan firebase-admin -python-paketti. Sen voi asentaa pip-paketin asentajalla käyttämällä seuraavaa komentoa [Google 2020l]:

```
pip install --upgrade firebase-admin
```

Asennettu paketti lisätään Python projektiin lisäämällä seuraavat moduulit koodiin:

```python
import firebase_admin
from firebase_admin import credentials
from firebase_admin import firestore
```

Palvelimella käyttöä varten tarvitaan JSON-tyyppiä olevan henkilökohtaisen avaimen, jonka voi luoda osoitteesta: https://console.cloud.google.com/iam-admin/serviceaccounts.

Sovelluksen alustamisessa annetaan polku henkilökohtaiselle avaimelle. Sovellus alustetaan seuravalla komennolla [Google 2020l]:

```
cred = credentials.Certificate('private-key.json')
firebase_admin.initialize_app(cred)

db = firestore.client()
```

Dataa tallennettaessa Firebastoreen, kokoelmia tai dokumentteja ei tarvitse luoda etukäteen, vaan Firestore luo ne itse tarvittaessa automaattisesti [Google 2020m]. Firestoren sallimia tietotyyppejä ovat merkkijonot (string), totuusarvomuuttujat (boolean), numerot (int, float), päivämäärät (datetime), tyhjät arvot (null), taulukot (array) sekä oliot (object) [Google 2020n].

Esimerkiksi seuraava koodi luo kokoelman ”sensors”, ja lisää sinne dokumentin ”tmpsensor001”, ja tmpsensor001:n alle tallennetaan set()-metodissa esitetyt avain-arvo -parit:

```
doc_ref = db.collection('sensors').document(' tmpsensor001')
doc_ref.set({
    'sensor': 'temperature',
    'sensorID': 'tmpsensor001'
})
```

Ensimmäisellä rivillä doc_ref-niminen muuttuja toimii viittauksena luotavalle dokumentille ”tmpsensor001”, joka tallennetaan sensors-kokoelmaan. Käytettävä tietokantametodi, kuten set(), kirjoitetaan aina kokoelman tai dokumentin viittauksen perään.

Vaihtoehtoisesti dokumentti voidaan jättää mainitsematta viittauksesta ja käyttää add()-metodia, joka generoi dokumentille ID:n, eli nimen, automaattisesti. Update()-metodia voidaan käyttää päivittämään dokumentissa olevia yksittäisiä kenttiä tarvitsematta korvata koko dokumenttia [Google 2020m]. Dokumentin hakemisessa tietokannasta käytetään vastaavasti viittausta dokumenttiin. Haku tapahtuu get()-metodilla [Google 2020o].

```
doc_ref = db.collection('sensors').document(' tmpsensor001')
doc = doc_ref.get()
```

Data saadaan doc-muuttujaan kokoelma-tietotyypissä (collection), jolloin haluttua arvoa voidaan käyttää viittaamalla sen kentän nimeen:

```
sensor_id = doc['sensorID']
```

where()-metodi mahdollistaa useamman dokumentin hakemisen kerrallaan. Where-kyselyfunktion avulla voidaan hakea dokumentit, jotka täyttävät tietyt ehdot. Tietokannasta voitaisiin hakea kaikki lämpötilasensorit seuraavalla komennolla [Google 2020o]:

```
docs = db.collection('sensors').where('sensor', '==', 'temperature').get()
```

Vertailu-operaatioita ovat perinteiset pienempi kuin, suurempi kuin ja samanarvoisuuden tarkistaminen. Lisäksi tarkasteluiden, kuten sijaitseeko arvo dokumentissa tai taulukossa, käyttö on myös mahdollista [Google 2020o].

Reaaliaikainen muutosten hakeminen tietokannasta suoritetaan kuuntelijalla. Halutulle dokumentille lisätään on_snapshot()-metodilla kuuntelija, joka muutoksia huomatessaan käynnistää on_snapshot-nimisen funktion, johon taas saadaan snapshotissa dokumentin muutokset. Kuuntelija voidaan vastaavasti kiinnittää kokonaiseen kokoelmaan [Google 2020p]:

```
def on_snapshot(doc_snapshot, changes, read_time):
    for change in changes:
        light = change.document.id
           print('new light-control snapshot for:', light)

doc_ref = db.collection('light_control')
doc_watch = doc_ref.on_snapshot(on_snapshot)
```

*changes*-parametrissa on tiedot dokumenteista, joihin on tehty muutoksia. *doc_snapshot* taas sisältää tässä tilanteessa koko light_control-kokoelman dokumentit, joista voidaan esimerkiksi hakea muutoksissa esitetty dokumentti.

Tarvittaessa kuuntelija voidaan myös irroittaa seuraavalla metodilla [Google 2020p]:

```
doc_watch.unsubscribe()
```

Myös dokumentin poistaminen Firestoresta onnistuu helpolla komennolla: kokonainen dokumentti voidaan poistaa tietokannasta käyttämällä delete()-

metodia dokumentin viittauksen perässä. Pelkän dokumentin sisäisen arvon poistaminen täytyy tehdä update()-metodin sisällä [Google 2020q]:

```
doc_ref.update({
    'measurement': firestore.DELETE_FIELD
})
```

Kokonaisen kokoelman poistaminen taas vaatii, että kokoelmasta poistetaan ensin kaikki dokumentit [Google 2020q].

### 4.2.4 Turvallisuussäännöt ja käyttäjän todentaminen

Firebase sovelluksissa voi käyttää käyttäjän todentamiseen salasanaa, puhelinnumeroa tai palveluiden, kuten Google, Facebook tai Twitter -tunnuksia [Google 2020r]. Firebasen turvallisuussäännöillä voi rajata mitä dataa kukin käyttäjä näkee tai voi muokata. Säännöt ovat käytettävissä reaaliaikatietokannassa, Cloud Firestoressa ja Cloud Storagessa [Google 2020s].

Firestoren säännöt voidaan kirjoittaa Firebasen konsolin kautta, tai sitten SDK:n avulla, jolloin säännöt julkaistaan esimerkiksi sovelluksen julkaisun yhteydessä.

Sääntöjä luodaan match-lauseilla, joiden avulla täsmätään tiettyyn Firestoren dokumenttiin. Match-lauseen alla annetaan haluttuja oikeuksia halutuin ehdoin. Oikeuksia voivat olla esimerkiksi luku, kirjoitus tai poistaminen [Google 2020t].

Ehtoina voivat olla esimerkiksi sisäänkirjautuminen tai dokumentin sisältämä ominaisuus. Esimerkiksi dokumenin omistajan ID voidaan tallentaa kyseisen dokumentin alle omistaja-kenttään, jolloin ehtoja tarkasteltaessa haun tehneen käyttäjän ID:tä verrataan tähän dokumentin alla olevaan kenttään. Esimerkiksi seuraavanlainen sääntö voisi antaa käyttäjän ohjata valoja jos käyttäjälle on asetettu oikeudet valoa ohjaavaan laitteen tietoihin:

```
match /device_control/{device}/lights/{light} {
   allow write: if request.auth.uid != null && exists(/databases/
   $(database)/documents/devices/$(device)/users/$(request.auth.uid));
}
```

Firestoren säännöt eivät koskaan päde dokumentin alla sijaitseville alikokoelman dokumenteille, vaan niille täytyy aina tehdä erikseen omat sääntönsä. Sääntöjen asettaminen sisäkkäin on kuitenkin mahdollista, jolloin alla olevan säännön polun täytyy ”jatkua” ylemmän match-lauseen polusta. Tosin säännöissä voidaan myös käyttää *jokerimerkkiä* (engl. wildcard), jonka avulla säännöt saadaan pätemään myös alikokoelmien dokumentteihin. Jokerimerkki lisätään match-lauseessa dokumentin perään [Google 2020t]:

```
match /dokumentit/{dokumentti=**)
```

## 4.3 Pilvifunktiot

Firebasen pilvessä toimivia funktioita voidaan käyttää automatisoimaan tehtäviä, jotka normaalisti suoritettaisiin palvelinkoodin toimesta. Funktiot voidaan asettaa käynnistymään esimerkiksi tietyn HTTP-pyynnön mukaan, tai jonkin Firebasen todennustapahtuman johdosta [Google 2020u].

Firebasen dokumentaatiossa havainnollistetaan pilvifunktioiden käyttöä käyttötapauksella, jossa sosiaalisen median sovelluksessa käyttäjälle annetaan ilmoitus hänen saadessaan uuden seuraajan. Seuraus-tapahtumaan liittyvän tiedon kirjoittaminen tietokantaan laukaisee pilvifunktion, joka lähettää seurattavalle henkilölle ilmoituksen Firebase Cloud Messaging (FCM) -toiminnon avulla [Google 2020v].

Mahdollisuus luoda automatisoituja tehtäviä pilvifunktioilla on IoT:n kannalta hyödyllistä. IoT-sovelluksessa funktioita voitaisiin käyttää antamaan käyttöliittymään ilmoituksia sensoridatan pohjalta: Käyttäjälle voitaisiin antaa ilmoitus tilanteessa, jossa liikeanturi havaitsee liikettä omistajan ollessa poissa tai jos lämpötila tai ilmankosteus nousee asetetun rajan yläpuolelle.

## 4.4 Yhteenveto

Ominaisuuksiensa puolesta Firebase on verrattavissa MongoDB Atlakseen; Firebasen monipuolisilla pilvitoiminnoilla voidaan useissa tapauksissa korvata erillisen palvelimen tarve. Firebase sopii myös erityisen hyvin prototyyppisovellusten tekemiseen sen tarjoaman ilmaisen maksusuunnitelman, helppokäyttöisyyden ja monialustatukensa vuoksi. Vaikka Firebase ei varsinaisesti ole kohdistettu IoT-sovellusten kehittämiseen, niin se on pilvipohjaisten ominaisuuksiensa vuoksi varteenotettava vaihtoehto: mahdollisuus datan säilömiseen ja prosessointiin pilvessä sekä sen reaaliaikainen päivittäminen sovellukseen sopivat IoT-tarkoituksiin erinomaisesti. Näillä ominaisuuksilla sitä siis voidaan käyttää ainakin osana IoT-alustaa.

Luvussa jäi mainitsematta vielä IoT-sovelluksille oleellisia tai ylipäätään sovelluskehityksessä hyödyllisiä toimintoja, kuten yksikkötestaaminen ja mahdollisuudet tiedon ulosvientiin palvelusta. Nämä siis on mahdollista toteuttaa Firebasen avulla, mutta tutkielmassa niitä ei käsitellä tarkemmin.

# 5 KÄYTTÖLIITTYMÄ JA TIEDON VISUALISOINTI

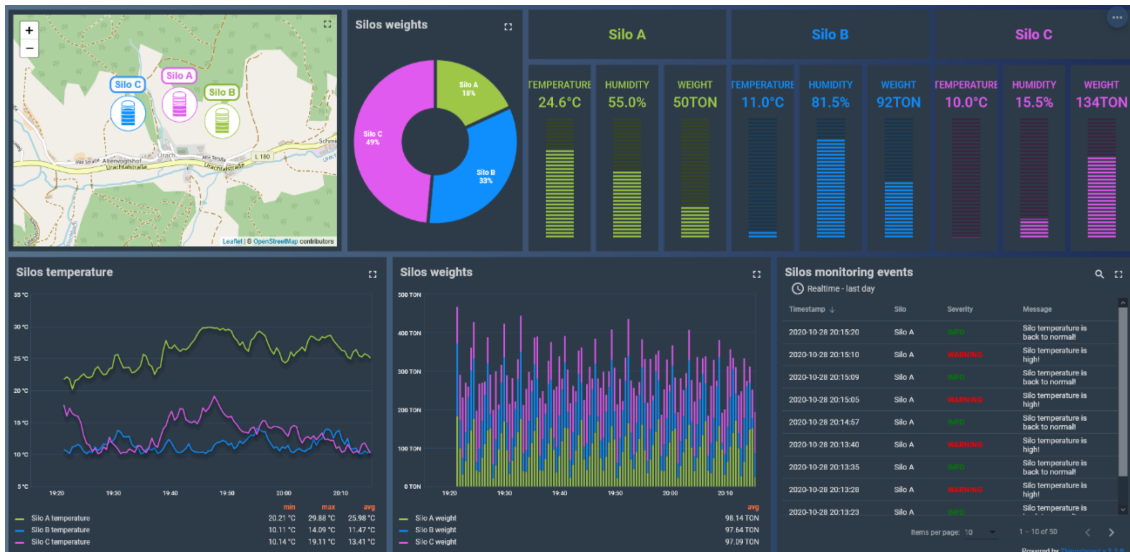
IoT-järjestelmä tarvitsee tavan esittää kerättyä tietoa sekä ohjata sen laitteita. Tässä kappaleessa esitän IoT-järjestelmän vaatimuksia käyttöliittymältä sekä käsittelen itse käyttöliittymän kehittämistä modernilla ohjelmistokehyksellä.

Vuorovaikutus IoT-järjestelmään toteutetaan käyttöliittymän avulla. Graafisella käyttöliittymällä voidaan luoda visuaalinen tapa käyttäjän ja sovelluksen väliseen vuorovaikutukseen: esimerkiksi web-sovellusta voidaan käyttää IoT-laitteen ohjaamiseen. IoT-sovelluksen käyttöliittymän kehittämisessä tärkeässä asemassa ovat sen passiiviset ominaisuudet, kuten hälytykset ja ilmoitukset. Myös oleellisia seikkoja ovat käyttäjätason mukaiset oikeudet sovelluksessa. Web-pohjaisen käyttöliittymän hyviä puolia ovat laiteriippumattomuus, ja se, että sen käyttäminen ei vaadi ohjelmistojen asentamista laitteelle. Toisaalta web-sovellus ei kykene esittämään mobiililaitteilla push-ilmoituksia oletuksena. Web-sovelluksen kehittämisessä on oleellista ottaa huomioon responsiivisuus sekä käyttäjäkokemus [Leverege 2020].

Käyttöliittymien kehittämiseen on olemassa valmiita IoT-ratkaisuja. Esimerkiksi thingsBoard on IoT-alusta, joka mahdollistaa datan keräämisen ja prosessoinnin lisäksi datan visualisoinnin ja kustomoitavat pienoisohjelmat, joita voidaan seurata kojelaudalta (engl. dashboard) [ThingsBoard 2020a].

Kuvassa 4 on ThingsBoardin verkkosivuilta löytyvä havainnollistava esimerkki käyttöliittymästä älymaatalouden käyttötapauksessa. Käyttöliittymästä voidaan seurata siilojen lämpötilaa, ilmankosteutta ja sisällön painoa. Mittauksista esitetään reaaliaikainen lukema sekä historiatiedot kaavioiden avulla. Lisäksi käyttöliittymässä on oma pienoisohjelmansa tapahtumien esittämiseen. Myös siilojen maantieteelliset sijainnit esitetään omassa pienoisohjelmassaan [ThingsBoard 2020b].

ThingsBoardin kojeulaudan esimerkki antaa hyvän kuvan siitä, mitä kaikkea IoT-järjestelmän käyttöliittymä voi tai tulisi sisältää. Pylväillä voidaan visualisoida eroja siilojen välisissä arvoissa ja väreillä voidaan havainnollistaa eri kuvaajissa sitä, mistä siilosta on kyse. Tapahtumissa värillä taas voidaan esittää tapahtuman vakavuutta. Kojelaudassa oleellista on myös, että kaikki pienoisohjelmat skaalautuvat ruudulle niin, että ruutua ei tarvitse koskaan vierittää. Käyttöliittymä on pidetty selkeänä esittämällä ainoastaan oleellinen informaatio. Lisäksi kojelaudan kustomoitavuus mahdollistaa käyttäjälle mieleisen kojelaudan ulkoasun toteuttamisen.




***Kuva 4.*** *ThingsBoardin esittelemä demo kojelaudasta [ThingsBoard 2020b].*

## 5.1 Firebase CLI & Firestore Web SDK

Luvussa 4 esitetyt Firestoren käyttöön liittyvät esimerkit pohjautuivat Firestoren Admin-ohjelmistokehityspakettiin Python-ohjelmointikielellä. Vaikka suurin osa Firestoren Web-ohjelmistokehityspaketin toiminnoista toimivat samalla periaatteella, niin tiettyjä eroavaisuuksia löytyy. Tässä kohdassa jatketaan hieman oleellisten Firestoren ominaisuuksien esittämistä asiakasohjelman osalta.

### Firebase CLI

*Firebase CLI*, eli Firebasen tarjoama komentoliittymä helpottaa Firebase-projektin hallintaa, mahdollista projektin testaamisen projektikoneella sekä sen julkaisemisen. Se on tarjolla Windows, macOS ja Linux -käyttöjärjestelmille. Firebase CLI:n voi asentaa *npm*-paketin hallitsijalla: https://docs.npmjs.com/about-npm/ [Google 2020x].

Seuraava komento asentaa Firebase CLI:n globaalisti käyttöjärjestelmässä:

```
npm install -g firebase-tools
```

CLI:n tärkeimmät komennot ovat:

```
firebase init projektin alustaminen projektikansiossa
firebase serve projektin emulointi lokaalisti
firebase deploy projektin julkaiseminen
```

### Firestore Web SDK

Kun Firestorea käytetään web-pohjaisessa asiakasohjelmassa, Cloud Firestore -kirjastot tulee lisätä sovellukseen samaan tapaan kuin Pythonilla ja sovellus tulee alustaa. Alustuksessa tarvitaan sovelluksen api-avain, käytettävä domain ja projektin ID [Google 2020y].

Paketti voidaan asentaa npm:llä:

```
npm install firebase@7.24.0 -save
```

save-parametri lisää paketin package.json-listaan. Package.json-listalle lisätään kaikki projektin vaatimat paketit, jotka myös voidaan ladata sen mukaisesti myöhemmin. Projektin alustaminen:

```
import firebase from 'firebase/app'
import 'firebase/auth'
import 'firebase/firestore'

var config = {
    apiKey: apiAvain,
    authDomain: domain,
    projectId: projektiId
};
firebase.initializeApp(config);
```

### Käyttäjän todentaminen

Käyttöliittymään käyttäjän sisäänkirjautuminen toteutetaan Firestoren auth-metodilla. Kirjautumisessa annetaan käyttäjän sähköposti sekä salasana [Google 2020z]:

```
firebase.auth().signInWithEmailAndPassword(email,password).catch(function(error) {
    …
```

## 5.2 Vue.js

Selainpohjainen käyttöliittymä soveltuu siis IoT-järjestelmän käyttämiseen hyvin. Tärkeimpiä vaatimuksia työkaluilta olivat helposti kehitettävät responsiiviset komponentit pienoisohjelmille sekä helppo tiedon päivittäminen käyttöliittymään. Useimmat modernit ohjelmistokehykset ovat ilmaisia käyttää sekä täyttävät vaadittavat ominaisuudet. Vue.js-ohjelmistokehys täytti nämät piirteet, ja käytinkin sitä apuna tutkimuksessani. Tässä kohdassa käsittelen sen perusteita.

Vue on käyttöliittymien kehittämiseen tehty ohjelmistokehys. Perinteisen MVC (engl. Model-view-controller) eli malli-näkymä-käsittelijä -arkkitehtuurin sijasta Vue perustuu vain näkymä tasoon. Jotta Vue.js:n ominaisuuksia voidaan käyttää sovelluksessa, sovellus tarvitsee Vue-instanssin [Vue.js 2020a]:

```
var app = new Vue({
    el: '#app',
    data: {
        sensor: 'Temperature'
    }
})
```

Vue käyttää HTML-pohjaisia *malleja* (engl. template) joiden avulla Vue-instanssin tietoa voidaan yhdistää DOM:iin, ja siten piirtää käyttöliittymään [Vue.js 2020b].

Vue-instanssi linkittyy haluttuun DOM-elementtiin *el*-ominaisuuden perusteella.

Vuessa tietoa voidaan kirjoittaa DOM:iin *deklaratiivisesti*: Vue-instanssin ominaisuuksia voidaan esittää mallin DOM-elementeissä käyttämällä aaltosulkumerkkejä. Tieto myös päivittyy DOM:iin automaattisesti Vue-sovelluksen datan muuttuessa. Tiedon näyttäminen elementin sisällä deklaratiivisesti [Vue.js 2020a]:

```
<div id="app">
  {{ sensor }}
</div>
```

Vuelle ominaista ovat myös sen tarjoamat *direktiivit* (engl. directive). Vuen direktiiveissä käytetään *v-* alkuista merkintää. Direktiivejä voidaan käyttää DOM-elementtien attribuuteissa. Esimerkiksi *v-if* direktiivillä voidaan instanssin ominaisuuden totuusarvon mukaan valita piirretäänkö elementti käyttöliittymään. *v-for* direktiivi taas mahdollistaa esimerkiksi elementtien luomisen listan pohjalta. v-on direktiiviä voidaan käyttää Vue-sovelluksen metodien kutsumiseen käyttöliittymästä [Vue.js 2020a].

Tärkeä Vuen ominaisuus on myös sen uudelleenkäytettävät komponentit. Vue-komponentti on käytännössä oma valmis Vue-instanssinsa, jota voidaan käyttää toisen komponentin mallissa eli templatessa. Komponentilla siis on oma pohjansa, ja sitä alustettaessa sille voidaan siirtää tietoa ylemmästä komponentista. Komponentti tarvitsee *props*-ominaisuuden, jotta sille voidaan siirtää tietoa *v-bind* direktiivin avulla [Vue.js 2020a].

Vue mahdollistaa toimintojen lisäämisen sovelluksen elinkaaren tapahtumiin. Esimerkiksi created-funktiot kutsutaan, kun Vue-instanssi luodaan [Vue.js 2020c].

**Vue CLI**

Vuelle on tarjolla oma kehitystä helpottava komentoliittymänsä. Vue CLI tarjoaa muun muassa komennot projektin luomiseen ja testaamiseen omalla tietokoneella. Oleellisin ominaisuus on kuitenkin projektin paketointi tuotantoa varten [Vue CLI 2020].

**Vue Router**

Vue router mahdollistaa yhden sivun sovelluksen luomisen komponenttipohjaisella navigoinnilla. Vue router tarvitsee oman instanssinsa. Navigoinnin luominen sovellukseen Vue routerin avulla toteutetaan seuraavalla tavalla [Vue Router 2020]:

- VueRouter-instanssille lisätään ominaisuus, jossa polut yhdistetään Vue-komponentteihin
- Luodaan oma DOM-elementti *Router-näkymälle*, johon piirretään routerin sillä hetkellä käyttämä komponentti
- *Router-linkeille* lisätään omat DOM-elementit komponenttiin

### 5.2.1 Kehittäminen

Tässä alakohdassa esitetään esimerkki IoT-järjestelmän käyttöliittymään mahdollisista vaatimuksista. Jatkona luvussa 3 esitetylle tapaukselle, samalle järjestelmälle tulisi kehittää käyttöliittymä, josta RPI:n keräämää dataa voidaan seurata, sekä sen laitteita ohjata.

**Vaatimukset**

Käyttöliittymän vaatimuksiin kuuluu käyttäjän todentaminen. Sisään kirjautuneen käyttäjän tulee nähdä IoT-yhdyskäytävät sekä niiden sensorit ja laitteet vain, jos käyttäjälle on asetettu niiden käyttöön oikeudet. Käyttäjän täytyy myös pystyä kirjautumaan ulos järjestelmästä.

Käyttöliittymään täytyy luoda omat pienoisohjelmansa kullekin sensorille ja laitteelle. Sensorin pienoisohjelmansa tulee näkyä reaaliaikainen mittauksen arvo. Liiketunnistimelta näytetään viimeksi havaitun liikkeen kellonaika sekä päivämäärä. Laitteen pienoisohjemassa täytyy olla myös oleelliset painikkeet laitteen ohjaamista varten. Lämpötilasensorilta näytetään nykyisen lämpötilan lisäksi mittausten keskiarvo vuorokaudelta, ylin ja alin mittaus sekä historiadata visualisoituna.

**Kehittäminen**

Käyttöliittymän kehittämiseen voidaan käyttää aiemmin esiteltyä Vue.js ohjelmistokehystä. Käyttäjän todentamiselle voidaan luoda oma sivunsa Vue-komponentilla. Komponenttiin luodaan oma metodinsa sisäänkirjautumiselle, jossa käytetään Firebasen auth()-metodia.

Myös kullekin laite- ja sensorityypille kannattaa luoda oma pienoisohjelmansa Vue-komponenteilla. Kaikki pienoisohjelmat näytetään samalla sivulla laitteet-komponentissa. Pienoisohjelmien komponentit luodaan Firestoresta haettujen laitteiden perusteella, ja niille siirretään ”propseissa” yhdyskäytävän tunniste, jota tarvitaan myös laitekohtaisen tiedon hakemisessa. Laitteiden hakeminen voidaan toteuttaa Vuen activated-funktiossa.

Kullekin laitteelle myös luodaan oman komponenttinsa Vue created-funktiossa Firestore kuuntelija, joka hakee reaaliaikaisesti laitekohtaisesta dokumentistaan uuden tiedon, ja näyttää sen käyttöliittymässä. Laitekohtaiset toiminnot, kuten ajan formatointi tai lämpötilojen keskiarvon laskeminen, kannattaa toteuttaa Vuen computed-ominaisuuksilla. Toiminnot, kuten kaavion päivittäminen, voidaan tehdä instanssin metodeissa.

Sovelluksessa voidaan käyttää Vuen *keep-alive* -komponenttia, joka mahdollistaa komponentin tietojen pitämisen välimuistissa, vaikka komponentti suljettaisiin. Tämä vähentää ylimääräisten tietokantakutsujen määrää. Keep-alive komponenttia voidaan käyttää sovelluksen ylimmällä tasolla, jolloin router-näkymän komponentti asetetaan sen alle.

Laitteiden ohjaamiseen käytettävä tieto voidaan tallentaa esimerkiksi omaan kokoelmaansa Firestoreen. Tällä vältytään mahdollisilta ongelmilta kuuntelijoiden kanssa, eikä laitteelta saatavan tiedon päälle kirjoittaminen ole hyvä käytäntö muutenkaan. Laitteen ohjaamiseen tarkoitetun dokumentin tulee sisältää ainakin laitteen tyyppi, tunniste ja ohjaukseen tarvittava informaatio. Valon kirkkauden säätö kannattaa toteuttaa portaittaisesti vähentämään tietokantakutsujen määrää.

Lämpötilan historiadatan visualisointiin kannattaa käyttää valmiita työkaluja jos halutaan säästää kehitykseen kuluvaa aikaa. Ilmaisia, tarkoitukseen sopivia työkaluja on tarjolla JavaScript-kirjastojen muodossa. Seuraavassa alakohdassa esitellään yksi vaihtoehto sekä käsitellään hieman visualisointia IoT-sovelluksissa.

### 5.2.2 Visualisointi

Jotta IoT-laitteiden keräämää tietoa voidaan hyödyntää, se täytyy muuttaa esitettävään, käyttötarkoituksen mukaiseen muotoon. Tiedon visualisointi on tärkeä osa tiedon hyödyntämisessä, mutta tiedon hyötykäyttö yleensä vaatii pelkän esittämisen lisäksi myös tiedon analysointia ja mahdollisesti yhdistämistä jo

olemassa olevaan dataan. Tavalla, miten tietoa osataan käyttää hyväksi, on yritystoiminnassa suuri merkitys [Linna 2020].

Käytettäville visualisoinnin työkaluille edellytyksinä on monitoroinnissa kyky reaaliaikaisen tiedon esittämiseen, ja valtavien tiedon määrien kanssa mahdollisuus yhdistää, suodattaa ja visualisoida tällaista dataa [Linna 2020].

Historiallisen datan visualisointiin voidaan käyttää esimerkiksi Chart.js-javascript kirjastoa.

Chart.js on suunnittelijoille ja kehittäjille tarkoitettu kirjasto kaavioiden luomiseen web-sovelluksissa. Kaavioiden piirtämiseen käytetään HTML5:n canvas elementtiä. Chart.js:n kaaviot skaalautuvat käyttöliittymässä myös responsiivisesti [Chart.js 2020a]. Chart.js voidaan asentaa seuraavalla NPM-komennolla [Chart.js 2020b]:

```
npm install chart.js –save
```

Kaaviolle täytyy luoda oma canvas-elementti HTML-dokumenttiin ja lisätä vaadittava kirjasto projektiin. Kaaviolle luodaan oma Chart-luokkansa, joka yhdistetään kaavion DOM-elementtiin. Chart-luokalle annetaan myös kaikki vaadittava tieto, kuten kaavion tyyppi, data ja kaavion asetukset [Chart.js 2020c].

Mittaukset asetetaan Chart-luokan data-taulukkoon ja ”nimikkeet” lisätään labels-taulukkoon. Kaavion päivittäminen tapahtuu lisäämällä tietoa samaisiin taulukoihin tai poistamalla sitä sieltä. Lopuksi täytyy kutsua Chart-luokan update()-metodia [Chart.js 2020d].

Sensoridatalle, jossa tietoon kuuluu mittaus ja aikaleima, voidaan käyttää viivakaaviota historiadatan visualisointiin. Viivakaavio saadaan asettamalla kaavion tyypiksi ’line’. Itse mittaukset asetetaan data-taulukkoon ja aikaleimat labels-taulukkoon. Kaavion asetuksista x-akselin tyypiksi kannattaa asettaa ’time’ ja jakaumaksi ’linear’, jolloin mittausten etäisyys toisistaan kaaviossa esitetään niiden ajan mukaan.

## 5.3 Yhteenveto

Tässä luvussa käsiteltiin lyhyesti käyttöliittymän vaatimuksia IoT-järjestelmän tapauksessa, sekä esitettiin ratkaisuja niiden täyttämiseen. Vue.js soveltui käyttöliittymän kehittämiseen IoT-tarkoituksessa hyvin. Komponentit helpottivat kojelaudan tyyppisen käyttöliittymän kehittämisessä huomattavasti. Jos järjestelmään lisätään uusia laitteita tai uusi yhdyskäytävä laitteineen, käyttöliittymään voidaan piirtää automaattisesti niille pienoisohjelmat ilman tarvetta tehdä käyttöliittymään muutoksia, kunhan mukana ei ole uusia laitetyyppejä. Toisaalta uudenlaisen laitteen lisääminen komponenttien avulla

sovellukseen on myös helppoa. Reaaliaikaisen tiedon päivittäminen DOM:iin, ja sen automaattinen käsittely onnistuu myös Vue:n avulla. Firebasen integroimisessa Vue-sovellukseen tai Firestore SDK:n metodien käyttämisessä siinä ei myöskään ilmennyt ongelmia.

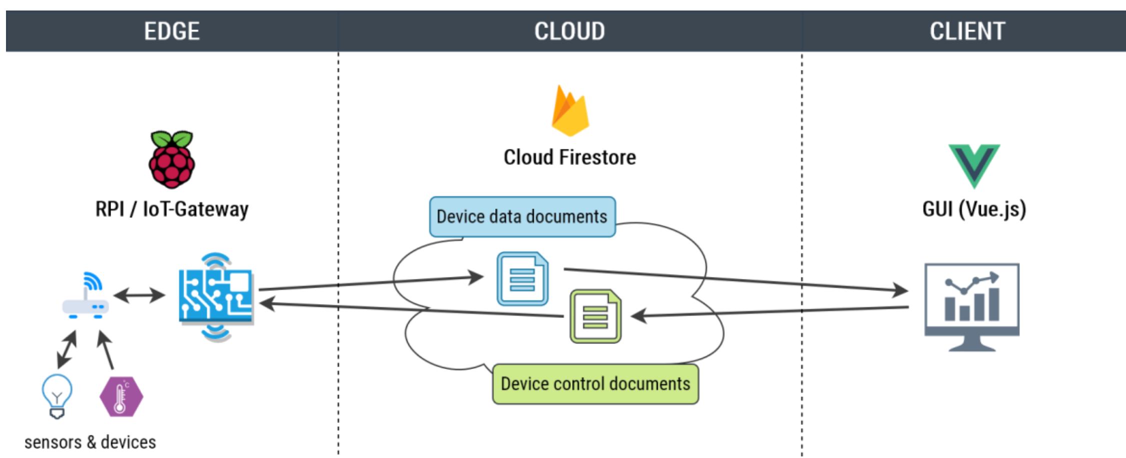
# 6 OMA TUTKIMUS JA TULOKSET

## 6.1 Järjestelmän kuvaus

Testiprojektina luotiin yksinkertainen IoT-järjestelmä, joka sisältäisi tutkimuksen pohjalta ilmenneistä IoT-järjestelmän piirteistä oleellisimmat. Tavoitteena oli saada paikallisen sensorin data pilveen, visualisoitua dataa reaaliaikaisesti käyttöliittymässä sekä myös pystyä ohjaamaan paikallista laitetta käyttöliittymästä. Toteutuksessa oli myös oleellista, että järjestelmä olisi helppo käyttöönottaa myös jälkikäteen ja että uusien laitteiden ja ominaisuuksien lisääminen järjestelmään tapahtuisi helposti. Tämä mahdollistettiin muun muassa ohjelmiston modulaarisuudella, konfiguraatiotiedostoja käyttämällä sekä versionhallinnalla. Ohjelmistojen ominaisuudet pidettiin mahdollisimman suppeina, jotta tutkimuksen kannalta oleellinen informaatio saataisiin pidettyä selkeästi esillä.

Kaikki projektissa käytettävät laitteet ja teknologiat olivat kohtuullisen helposti saatavilla. Projektissani käytin Raspberry Pi:tä IoT-yhdyskäytävänä sensoreiden ja IoT-alustan välillä. Dataa keräsin Philips Hue -laitteisiin sisältyviltä antureilta. IoT-alustana toimi Firebase ja tietokantana Cloud Firestore. Käyttöliittymänä toimi Vue.js JavaScript-ohjelmistokehyksellä luotu web-sovellus.

Kuva 5 esittää kehitetyn järjestelmän rakenteen. Järjestelmän reunalla RPI suorittaa kehittämääni ohjelmistoa, joka mahdollistaa soveltuvien lähiverkon laitteiden ohjaamisen sekä niiltä datan keräämisen. Data haetaan sensoreilta REST API:n avulla, muutetaan soveltuvaksi ja lähetetään Cloud Firestore -tietokantaan. Tiedon päivittyessä tietokannassa tieto synkronoidaan reaaliajassa myös mahdollisille asiakasohjelmille. Käyttöliittymästä voidaan seurata dataa reaaliajassa sekä ohjata laitteita. Käyttöliittymän kautta suoritettu ohjaaminen kirjoittaa asetukset Firestore-tietokantaan, jolloin RPI saa reaaliajassa tiedon uusista asetuksista ja ohjaa lähiverkon laitteita näiden mukaisesti.




***Kuva 5.*** *Toteutetun IoT-järjestelmän tasot.*

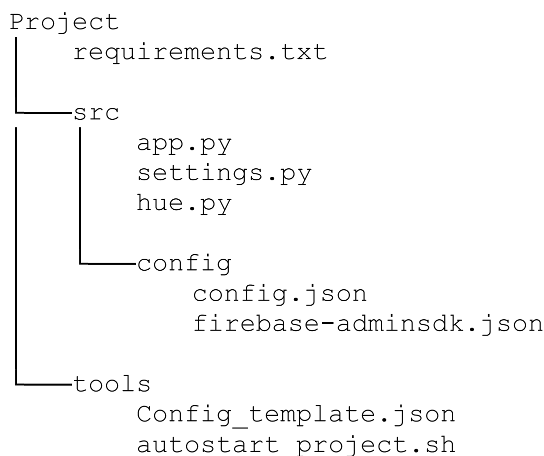
### 6.1.1 IoT-laitteet

Käytössäni oli aiemmin esitelty Raspberry Pi 3 Model B, jonka käyttöjärjestelmänä toimi Raspberry Pi OS Lite. Raspberry Pi oli kytkettynä lähiverkkooni ethernet-kaapelilla. Raspberry Pi toimi projektissani IoT-yhdyskäytävänä keräämällä ja prosessoimalla dataa, ohjaamalla lähiverkon laitteita sekä lähettämällä dataa IoT-alustalle pilveen. Ohjattavia IoT-laitteita lähiverkossa ovat Hue-valaisimet sekä Hue Smart plug -etäohjattava pistorasia. Sensoreina toimivat Hue-liikkeentunnistimen lämpötila, valoisuus sekä liiketunnistin-anturit. Philips Huen tarjoama RESTful API mahdollisti helpon, valmiin ratkaisun kommunikaatioon laitteiden välille.

Ohjelmointikielenä käytin Pythonia sen helppokäyttöisyyden sekä sille tarjolla olevien datatiedekirjastojen vuoksi. Firestoren tarjoamista ohjelmistokehityspaketeista ainoastaan Python sekä Node.js olivat saatavilla Raspberry Pi OS:lle. Python oli minulle myös ennestään varsin tuttu ohjelmointikieli, joten myös sillä kehittämisen aloittaminen oli helppoa.

Projektissa käytin Git-versionhallintaa. Ideana oli, että projektin voisi helposti hakea myös toiselle RPI:lle suoraan gitistä ja lataamalla requirements.txt:n mukaiset paketit, sekä täyttämällä projektin sisäisen konfiguraatiotiedoston laite olisi heti valmis yhdistettäväksi järjestelmään.

Python-projektin kansiorakenne:



```
Project
│   requirements.txt
│
└───src
│   │   app.py
│   │   settings.py
│   │   hue.py
│   │
│   └───config
│           config.json
│           firebase-adminsdk.json
│
└───tools
        Config_template.json
        autostart_project.sh
```

Projektille oli luotu oma python-virtuaaliympäristönsä. App.py on projektin päätiedosto, josta sovellus käynnistetään. App.py kattaa suurimman osan ohjelman toiminnallisuudesta. Kaikki Hue API -kommunikaatio on luotu hue.py-tiedostoon: Huelle luotiin oma luokkansa, jolla on metodit valojen ja laitteiden tietojen hakemiseen sekä valojen ohjaamiseen. Config-kansion alta löytyy Firestoren vaatima service-account -avain ja laitekohtainen konfiguraatiotiedosto. Tools-kansiosta löytyy skripti, jolla ohjelma saadaan Raspberry Pin käynnistymisen mukana käynnistymään automaattisesti.

JSON-muotoiseen konfiguraatiotiedostoon täytetään ohjelman kannalta oleellista tietoa, kuten Firebase-projektin ID, laitteelle käyttäjä ja tiedot Hue API:n käyttöä varten:

```
{
    "device":{
        "id":"1",
        "name":"RasPi1",
        "desc":"Device Description"
    },
    "firebase":{
        "project_id":"project-id",
        "certificate_file":"config/adminsdk-certfile.json",
        "users": [
            {"uid":"user-uid-from-firebase", "permission": "admin"}
        ]
    },
    "hue":{
        "api_user":"hue-api-user",
        "bridge_ip":"192.168.1.44"
    }
}
```

Sovelluksen käynnistyessä konfiguraatiotiedostosta luetaan ja päivitetään Firestore-tietokantaan sovellusta pyörittävän RPI:n tiedot sekä sallitut käyttäjät. Konfiguraatioiden jälkeen haetaan sensorit Hue API:lta, ja ne päivitetään aina Firestoreen mahdollisten uusien laitteiden varalta. Myös Hue-älyvalot päivitetään. Laitteen tyypin mukaiseen kokoelmaan laitteelle luodaan oma dokumenttinsa, josta löytyy sille oleellisimmat tiedot. Dokumentin nimeksi annetaan laitteen nimi selkeyden vuoksi.

Lämpötilasensorilta tallennettavat tiedot, niiden Python-tyypit ja selitteet:

| | | |
|---|---|---|
| **id** | string | sensorin tunniste |
| **name** | string | laitteen nimi |
| **reachable** | boolean | laitteen tavoitettavuus |
| **temp** | float | sensorilta saatu lämpötila |
| **ts** | datetime | mittauksen ajankohta |
| **type** | string | sensorin tyyppi |

Kuva 8 esittää lämpötilasensorin tiedot Firestore-tietokannassa. Liiketunnistimelta tallennetaan muuten vastaavat tiedot, mutta lämpötilan sijasta tietokantaan lähetetään tieto havaitusta liikkeestä:

```
presence          boolean       onko liikettä havaittu
```

Sensorin dokumenttiin siis tallennetaan sensorin oleellisimpien tietojen lisäksi viimeisimmän mittauksen lämpötila ja ajankohta. Tämän lisäksi sensorilla on Firestoressa oma data-kokoelmansa, johon lämpötila ja ajankohta tallennetaan omaan dokumenttiinsa. Data-kokoelmaan tallennettavaa dataa käytetään visualisointiin, eikä sitä koskaan ylikirjoiteta.

Älyvalolta tallennettavat tiedot id:n, nimen, tavoitettavuuden ja tyypin lisäksi:

```
bri               int           valon kirkkaus (arvo väliltä 0-254)
on                boolean       valon päällekytkennän tila
```

Sensoreiden hakemisen jälkeen sovellus käynnistää omat säikeensä kullekin sensorille tiedon hakemista varten. Säikeet toteutettiin threading-moduulin avulla. Säikeiden avulla sensoridataa voidaan päivittää tietokantaan eri tahtiin eri sensoreilla Firestoresta saatujen asetusten esittämän päivitystiheyden mukaan. Sensorityypeille tarvittiin myös omanlaisensa funktiot: liiketunnistimelta halutaan lähettää dataa ainoastaan, kun liikettä havaitaan (tai kun liike lakkaa), kun taas lämpötilaa halutaan mitata aina tietyin väliajoin. Valoille luotiin oma kuuntelijansa niiden ohjaamista varten, jotta käyttöliittymän kautta tehdyt säädöt saadaan asetettua valoille reaaliajassa. Kehityksen alkuvaiheilla tosin ilmeni virhe Firestoren Python SDK:ssa, jossa pitkään käynnissä ollut kuuntelija kaatuu ja lakkaa toimimasta tietyn ajan jälkeen. Tämän ongelma saatiin ohitettua sillä, että kuuntelijalle luotiin oma säikeensä, joka sammutetaan ja käynnistetään uusiksi esimerkiksi 30:n minuutin välein. Säikeiden pyöriessä taustalla itse pääprosessi saatiin pidettyä käynnissä signal-kirjaston pause()-metodilla.

Esimerkki lämpötilasensorilta tiedon hakemiseen sekä tallentamiseen tehdystä pelkistetystä tehtävästä, joka suoritetaan omassa säikeessään:

```
def temperature_task(sensor):
    try:
        print("Creating temperature task for", sensor["name"])
        while True:
            print("timed update:", sensor["name"], datetime.datetime.now())
            new_data = hue.get_sensor(sensor["id"])
            temp = new_data["state"]["temperature"] / 100
            reachable = new_data["config"]["reachable"]
            ts = new_data["state"]["lastupdated"]
            sensors_ref.document(sensor["name"]).update({"temp": temp,
                                                         "reachable": reachable,
                                                         "ts": ts})
            if reachable:
                sensors_ref.document(sensor["name"]).
                    collection("data").document(ts).set({"temp": temp,
                                                         "ts": ts})
            else:
                print(sensor["name"], "not reachable")
            sleep(sensor["update_interval"])
    except KeyboardInterrupt:
        print("temperature task ended for sensor:", sensor["name"])
        pass
```

Helppokäyttöisyyttä tukemaan ohjelmisto laitettiin käynnistymään taustaprosessiksi automaattisesti Raspberry Pi:n käynnistyessä.

### 6.1.2 IoT-alusta

Firebasen tietokannoista käytin projektissani Cloud Firestorea sen monipuolisuutensa ansiosta. Tietokannan valinnassa halusin ottaa huomioon IoT:n kannalta oleellisia seikkoja, kuten saatavuuden, monipuoliset mahdollisuudet kyselyille ja skaalautuvuuden, vaikka ne eivät omassa projektissani selkeästi tulleetkaan esille sen pienen koon johdosta. Firestoren monipuolisemmat ominaisuudet kuitenkin vaikuttivat kiinnostavilta.

Järjestelmän käyttäjät luotiin Firebase konsolin kautta. Tietokannan säännöt toteutettiin laitekohtaisesti: jos laitteen käyttäjät-kokoelmasta löytyy dokumentti käyttäjän id:llä, niin hänellä on oikeudet sääntöjen määräämiin operaatioihin. Koska käyttäjien ei ole tarkoitus voida kirjoittaa saadun datan yli, niin ”devices” dokumenteille on aina pelkästään lukuoikeudet. Ainoastaan ”device_control”-kokoelman alla oleviin dokumentteihin on tarkoitus voida kirjoittaa käyttäjien toimesta.

Kuva 6. Sensorin tiedot Firestore-tietokannassa [Google 2020ä].

***Kuva 6.*** *Sensorin tiedot Firestore-tietokannassa [Google 2020ä].*

Projektille tehdyt Firestoren turvallisuussäännöt:

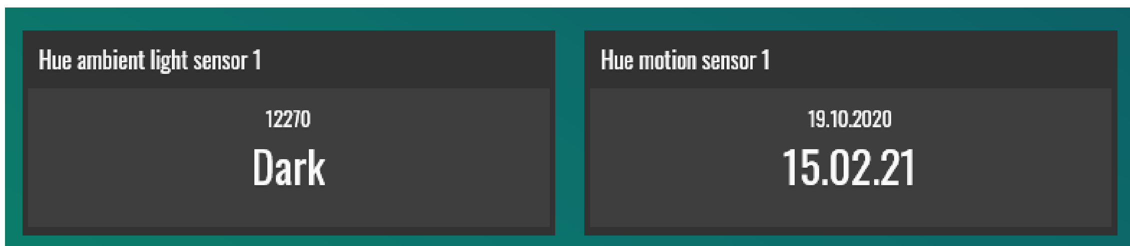
```
service cloud.firestore {
  match /databases/{database}/documents {
    match /devices/{device}/lights/{light} {
      allow read: if request.auth.uid != null && exists(/databases/
      $(database)/documents/devices/$(device)/users/$(request.auth.uid));
    }
    match /devices/{device}/sensors/{sensor} {
      allow read: if request.auth.uid != null && exists(/databases/
      $(database)/documents/devices/$(device)/users/$(request.auth.uid));
      match /data/{data} {
        allow read: if request.auth.uid != null && exists(/databases/$
        (database)/documents/devices/$(device)/users/$(request.auth.uid));
      }
    }
    match /device_control/{device}/lights/{light} {
      allow write: if request.auth.uid != null && exists(/databases/
      $(database)/documents/devices/$(device)/users/$(request.auth.uid));
    }
    match /users/{userId} {
      allow read: if request.auth.uid != null &&
      request.auth.uid == userId;
    }
  }
}
```

### 6.1.3 Käyttöliittymä

Käyttöliittymä kehitettiin Vue.js JavaScript-ohjelmistokehystä apuna käyttäen perinteisten web-kehityksessä käytettävien teknologioiden lisäksi. Käyttöliittymä asennettiin Firebasen hosting-palveluun.

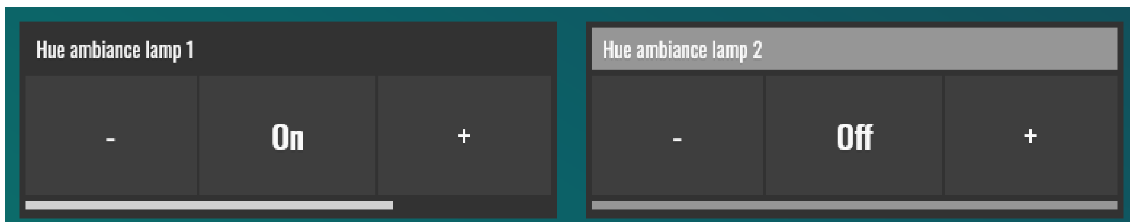
Käyttöliittymän käyttäminen vaatii käyttäjätunnuksilla kirjautumisen. Käyttäjätunnusten luominen tehtiin Firebase konsolin kautta. Kirjautuneen käyttäjän tietoja voitiin käyttää oikean tiedon hakemiseen tietokannasta. Sovellusta aukaistaessa käyttäjä ohjataan automaattisesti kirjautumisen sivulle. Sisäänkirjautumisen jälkeen käyttäjä ohjataan sivulle, josta löytyy käyttäjälle sallitut yhdyskäytävät. Yhdyskäytävän valitsemalla käyttäjälle esitetään yhdyskäytäväkohtainen kojelauta, jossa esitetään pienoisohjelmat sen sensoreille ja laitteille.

Käyttöliittymään luodaan jokaiselle Firestoresta haetulle laitteelle oman Vue-komponenttinsa, jossa esitetään laitteen tyypin mukaan oleellinen data, ja mahdolliset painikkeet ohjaamiselle. Kuva 7 esittää valoisuusanturille tehdyn komponentin, josta ilmenee päivänvalon tila sekä kirkkaus. Liikeanturin komponentista taas selviää viimemmäksi havaitun liikkeen kellonaika sekä päivämäärä.




***Kuva 7.*** *Käyttöliittymään kehitetyt pienoisohjelmat sensoreille.*

Valon komponentti on esitetty kuvassa 8. Komponentista ilmenee valon päällekytkentä sekä kirkkaus. Komponentissa on myös painikkeet valon kirkkauden säätämiselle sekä sen sammuttamiselle tai käynnistämiselle.




***Kuva 8.*** *Käyttöliittymään kehitetty pienoisohjelma valoille.*

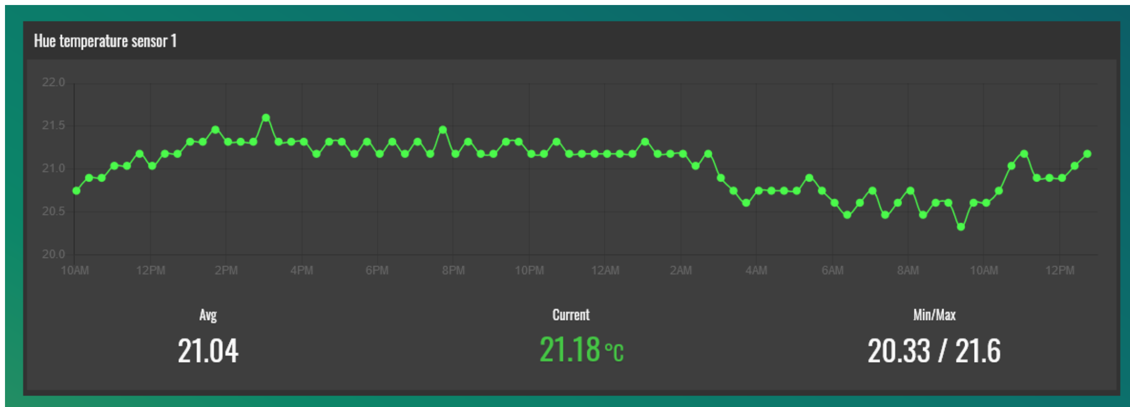
Esimerkkinä valolle kehitetyn Vue-komponentin malli:

```
<template>
    <div v-if="light.type !== 'On/Off plug-in unit'" class="light-container">
        <div :class="[!light.on || !light.reachable ? 'light__disabled' : '']"
              class="header">
          {{ light.name }}
        </div>
        <div v-if="light.reachable">
          <div class="content noselect">
              <div class="btn minus-btn" @click="decreaseBrightness"> - </div>
              <div class="btn toggle-btn" @click="toggleLight"> {{ statusToText }} </div>
              <div class="btn plus-btn" @click="increaseBrightness"> + </div>
          </div>
          <div class="brightness-container">
            <div class="brightness-bar" v-bind:style="{ width: briWidth }"
                                         :class="[light.on ? '' : 'light__disabled']">
            </div>
          </div>
        </div>
        <div v-else>
          <div class="offline noselect">OFFLINE</div>
        </div>
    </div>
</template>
```

Valon kirkkautta visualisoitiin painikkeiden alla olevalla vaakasuoralla pylväällä. Sensorilta saadun arvon muuttaminen prosentuaaliseksi leveydeksi CSS-tyylimäärityksiä varten voitiin toteuttaa Vuen computed-funktion avulla:

```
computed: {
  briWidth: function() {
    return ((this.light.bri / 254) * 100).toFixed(0).toString() + "%"
  }
```

Historiallisen lämpötiladatan visualisointiin käytettiin viivakaaviota. Kaavio on esitetty kuvassa 9. Viivakaavio luotiin Chart.js JavaScript-kirjastoa apuna käyttäen.




***Kuva 9.*** *Käyttöliittymään kehitetty pienoisohjelma lämpötiladatalle.*

Kaavio esittää datan viimeiseltä vuorokaudelta. Kaavion alla vasemmalla esitetään lämpötilan keskiarvo, keskellä esitetään nykyinen lämpötila ja oikealla alin ja korkein lukema. Kuvan tilanteessa on käytetty 20 minuutin intervallia näytteenottoon. Sensoridata viimeiseltä 24 tunnilta saatiin käyttämällä Firestoren tietokantakysely-lauseita. ”Where”-lauseessa valitaan kokoelmasta dokumentit, joiden aikaleima (ts) on viimeisen 24 tunnin sisältä. ”orderBy”-lauseella data saatiin nousevaan järjestykseen aikaleiman mukaan kaaviota varten.

```
var last_day = new Date();
last_day.setHours(last_day.getHours() - 24);

this.sensor_data_ref
.where('ts', '>', last_day)
.orderBy("ts", "asc").get().then(function(querySnapshot) {
    querySnapshot.forEach(function(doc) {
        var data = doc.data();
        …
```

Kun laitteen sivu avataan käyttöliittymästä, sovellus ensiksi hakee 24 tunnin datan Firestoresta. Tämä toteutetaan Vuen mounted-funktiossa. Komponenttikohtainen mounted-funktio kutsutaan aina kerran kun komponentin instanssi luodaan. Vuen created-funktiossa taas lisätään kuuntelija, joka hakee vain tuoreimman, reaaliaikaisen mittauksen Firestoresta ja lisää sen myös aiemman datan perään kaavioon. Tällä vältyttiin suuren datamäärän hakemiselta aina tiedon päivittyessä.

Kaikki käyttöliittymästä tehdyt komennot kirjoitetaan Firestoren device_control-kokoelmaan, josta Raspberry Pi vastaavasti lukee ja toteuttaa komennot.

### 6.1.4 Testit

Raspberry Pi:llä suoritettavan sovelluksen muistin ja prosessorin käyttöä pystyi seuraamaan komentorivin *top*-komennolla, joka kertoo prosessikohtaiset käytöt reaaliajassa. Sovelluksen muistinkäyttä oli keskimäärin 5.5% juuri vaihtelematta, ja prosesorinkäyttö keskimäärin 2%, sekä korkeintaan 9%. Prosessorin lämpötila oli keskimäärin 45°C.

Järjestelmän tehokkuutta voitiin mitata erilaisilla testeillä. Ensimmäisessä testissä Python sovellus lähettää dataa Firestoren dokumenttiin ja tallentaa lähetyksen ajankohdan. Samassa sovelluksessa on myös kuuntelija, joka hakee kyseisestä dokumentista saman tiedon heti ja ilmoittaa haun ajankohdan. Viive saatiin laskemalla ajankohtien erotus. Taulukko 1 esittää testin viiveen keskiarvon, minimi- sekä maksimilukemat.

| Näytteet | Keskiarvo | Min | Max |
|---|---|---|---|
| 15 | ~134ms | 100ms | 166ms |

***Taulukko 1.*** *Tiedon kulku RPI:ltä Firestoreen ja takaisin kuuntelijoilla.*

Toisessa testissä kokeilin, kuinka tiheään kuuntelija voi ottaa vastaan uusia snapshoteja. Testissä käyttöliittymän kautta lähetettiin mahdollisimman tiheään ohjauskomentoja Firestoreen, ja RPI:llä suoritettavan ohjelman kuuntelija tallensi saamiensa snapshotien ajankohdat. Peräkkäisten snapshotien välinen viive laskettiin. Taulukko 2 esittää toisen testin tulokset.

| Näytteet | Keskiarvo | Min | Max |
|---|---|---|---|
| 15 | ~777ms | 713ms | 833ms |

***Taulukko 2.*** *Kuuntelijan kuormittaminen.*

Kuuntelija siis kykeni hakemaan tietoa korkeintaan noin 0,7-0,8 sekunnin välein kuormituksen alaisena. Testeissä on hyvä huomioida ainakin asiakasohjelman sekä oman Internet-yhteyden vaikutus viiveeseen. Mittausten aikana speedtest.net:in mukaiset nopeudet sekä selaimen tiedot:

**PING:** 5ms
**Lataus:** 91.76 Mbps
**Ylöslataus:** 10.38 Mbps
**Selain:** Firefox v. 82.0.2 (64-bit)

## 6.2 Lopputulokset

Lopputuloksena saatiin kehitettyä yksinkertainen, mutta toimiva IoT-järjestelmä, joka täytti sille itse asettamani vaatimukset. Järjestelmän ei ollut koskaan tarkoitus olla verrattavissa todelliseen IoT-järjestelmään täydellisten ratkaisujen puitteissa, vaan se oli ennemminkin *pienin toimiva tuote*, jolla saatiin kodin älylaitteet esille yhteiseen järjestelmään ja jossa voitiin käyttäjäkohtaisten oikeuksien mukaisesti seurata dataa sekä ohjata järjestelmän laitteita. Kehityksen aikana sain selkeämmän kuvan siitä, mitä olisi kannattanut tehdä toisin ja mitkä valinnat taas toimivat erityisen hyvin toteutuksen kannalta sekä minkälaisiin tarkoituksiin järjestelmää voisi jatkokehityksen myötä käyttää. Järjestelmän hinnaksi tuli Raspberry Pin hinta eli ~45e + sensorit ja laitteet. Itse Hue-laitteet eivät ole markkinoiden halvin vaihtoehto, mutta ainakin ne tarjoavat helpon kehitettävyyden.

Raspberry Pi täytti hyvin yhdyskäytävän roolin projektissa. Sille saatiin kehitettyä soveltuva ohjelmisto laitteiden sekä Firestoren kanssa kommunikointia varten. Hintansa ja ominaisuuksiensa puolesta RPI oli varma valinta ohjelmiston prototyypin kehittämiseen. Lisäksi Raspberry Pi OS on erinomainen ja varma käyttöjärjestelmä kehittäjille sen tukensa vuoksi. Ohjelmiston kehittämiseen Python-ohjelmointikieli toimi riittävän hyvin, mutta uskon, että Node.js olisi ollut parempi ratkaisu, etenkin käyttöön Firestoren kanssa. Node.js on ohjelmiston kannalta huomattavasti nopeampi ja tehokkaampi [Costa 2020]. Lisäksi sen käyttäminen olisi helpottanut kehitysprosessia mahdollistamalla JavaScript-ohjelmointikielen käyttämisen kaikkialla järjestelmässä.

Firebase sopi tarkoitukseeni IoT-alustana kohtalaisen hyvin, koska käytössäni oli vain pieni määrä sensoreita ja muita laitteilta, joilta dataa tallennetaan pilveen. Normaalissa käytössä tietokantaan kirjoituksia tuli alle 5% vuorokauden kiintiöstä sekä lukuja alle 2%. Suurten sensorimäärien kanssa Firestoren maksusuunnitelma voisi tulla IoT-sovelluksissa kalliiksi. Myöskään sen nopeuden ei voi olettaa olla samalla tasolla IoT-tarkoituksiin erikoistuneiden alustojen kanssa. Oman järjestelmäni käyttötarkoituksessa eli älykodin merkeissä nopeus tai päivitystiheys ei ollut kuitenkaan tärkeintä, eikä toiminnanvarmuudella ollut kriittistä merkitystä. Firebase voi siis olla yksinäänkin hyvä vaihtoehto, jos järjestelmällä ei ole useita käyttäjiä. Suosittelisin kuitenkin RTDB:tä Firestoren sijaan IoT-käytössä.

Tutkimuksessa oli alun perin tarkoitus kokeilla myös pilvifunktioita sekä koneoppimista, mutta ne eivät lopulta sopineet aiheeseeni. Pilvifunktioille olisi IoT-sovelluksissa selvät käyttökohteet, erityisesti ilmoituksien tekemisessä. Koneoppimisen käyttäminen taas olisi vaatinut selvän käyttötarkoituksen ja todennäköisesti dataa useamman tyyppisiltä sensoreilta.

Valmiisiin vaihtoehtoihin verrattaessa oman järjestelmän kehittäminen mahdollistaa vapaan käytettävyyden kaikille Huen RESTful API:n tarjoamille ominaisuuksille. API:n avulla laitteista voi myös saada enemmän irti: esimerkiksi Hue-liiketunnistimessa olevien valoisuusanturin ja lämpötilasensorin lukemat eivät olleet nähtävillä ilman API:a. Oman järjestelmän kehittäminen mahdollistaa myös useiden eri valmistajien laitteiden yhdistämisen samaan järjestelmään ja kommunikaation luomisen niiden välille, tietysti niiden tarjoamien rajapintojen puitteissa.

## 6.3 Jatkokehitys

Ilmoitukset voisi luoda käyttöliittymään Firebasen pilvifunktioilla. Jos projekti sisältäisi suuremman prioriteetin ilmoituksia, niin silloin mobiilisovellus olisi hyvä lisä järjestelmään. Myös tietoa olisi hyvä pystyä tuomaan ulos järjestelmästä, esimerkiksi valittujen päivämäärien mukaan. Tiedon ulosviennille voisi toteuttaa oman toimintonsa käyttöliittymään. Käyttöliittymä voisi myös mahdollistaa kojelaudan käyttäjäkohtaisen kustomoitavuuden ja tietynlaisten toimintojen yhteenliittämisen. Esimerkiksi kojelautaan voisi luoda oman painikkeen kytkemään useiden laitteiden halutut ominaisuudet kerralla. Lisäksi käyttöliittymästä voitaisiin luoda tapahtumia yhdistelemällä laitteiden ominaisuuksia: esimerkiksi, jos tietty lämpötilasensori ylittää tietyn arvon, niin jokin haluttu järjestelmän laite kytketään päälle.

Järjestelmään voisi kokeilla yhdistää useampia yhdyskäytäviä laitteineen. Yhdyskäytävien ja laitteiden väliseen kommunikaatioon voisi myös kehittää omat toimintonsa. Firebasen tietokannat sopivat hyvin tiedon näyttämiseen asiakasohjelmissa, mutta en usko sen olevan oikea ratkaisu kommunikaatioon yhdyskäytävien kanssa. Apuna voisi hyödyntää muita Google Cloudin IoT-palveluita. Uskon, että *Cloud IoT Core* MQTT-protokollan kanssa soveltuisi hyvin tarkoitukseen.

Yhdyskäytävän käyttöönottoa sekä muita mahdollisia ominaisuuksia helpottamaan yhdyskäytävälle voisi toteuttaa paikallisessa verkossa toimivan käyttöliittymän. Tämä kuitenkin vaatisi yhdyskäytävän käyttämisen paikallisena palvelimena, jolloin ohjelmisto kannattaisi toteuttaa uudelleen esimerkiksi Node.js:llä tai vaihtoehtoisesti käyttää *Flaskia*.

Flask on web-sovellusten kehittämiseen tarkoitettu kevyt ohjelmistokehys, joka käyttää Python-ohjelmointikieltä [Pallets Projects 2020].

Sensoridatan analysointiin voisi käyttää koneoppimista. Pilvessä koneoppimiseen voisi käyttää Google Cloudin Cloud ML Engineä, ja vaihtoehtoisesti Raspberry Pi:llä voitaisiin käyttää *Tensorflowta*.

Tensorflow on avoimeen lähdekoodiin pohjautuva koneoppimisalusta [Tensorflow 2020].

Vaihtoehtona älykodille työpaikoilla tai opetustiloissa voitaisiin myös käyttää toteutetun järjestelmän tapaista, käyttäjäoikeuksiin pohjautuvaa järjestelmää. Esimerkiksi vierailijoilla voisi olla perusoikeudet ulko- ja sisälämpötilan tai muiden tietojen seuraamiseen, kun taas työntekijällä tai opiskelijalla olisi lisäksi oikeudet valojen ja muiden laitteiden säätämiseen. Samaan järjestelmään olisi mahdollista integroida muita rakennuksen tai sijainnin tietoja, kuten aukioloaikoja, ohjeistusta ja karttoja. Älykodissa tai muissa tapauksissa järjestelmä voisi hyödyntää käyttäjien älypuhelinten sensoreita järjestelmän lisälaitteina niille soveltuvissa tarkoituksissa. Tämä voitaisiin toteuttaa *Generic Sensor API*:n avulla.

Generic Sensor API:n rajapinnat mahdollistavat esimerkiksi älypuhelimen sensoreiden datan käyttämisen web-alustoilla [Pozdnyakov ja Shalamov 2020].

Toki tällaisten järjestelmien kehittämisessä tulee huomioida käyttäjän yksityisyys sekä järjestelmän turvallisuus perusteellisemmin. Vierailija voitaisiin yhdistää järjestelmään jonkin passiivisesti jäljitettävän laitteen avulla. Vaihtoehtoisesti yhdyskäytävään voisi olla yhdistettynä näyttö, jossa esitetään esimerkiksi QR-koodi. QR-koodin skannaamalla sovellukseen käyttäjälle annettaisiin perusoikeudet järjestelmään. QR-koodi voitaisiin käyttää vain kerran, ja se vaihtuisi näytöllä heti käyttäjän tunnistautumisen jälkeen. Tällaisen tunnistautumisen avulla voitaisiin myös rajoittaa aikaa, jona käyttäjällä on oikeudet järjestelmään. Oikeuksien loppuessa käyttäjän täytyisi skannata uusi koodi näytöltä.

Järjestelmä ei siis vaatisi vierailijalta mobiilisovelluksien tai käyttäjätunnuksien luomista, vaan voisi esimerkiksi tunnuksen tai koodin avulla siirtyä yhdyskäytävä- tai rakennuskohtaiselle web-sivulle, josta hän voisi seurata vierailijalle sallittavia tietoja sekä mahdollisesti käyttää laitteiden ohjaamiseen liittyviä toimintoja. Tällä tavalla voitaisiin luoda helppokäyttöinen järjestelmä, joka mahdollistaisi sen toimintojen käytön myös vierailijoille yksinkertaisilla toimenpiteillä.

# 7 LÄHDELUETTELO

Adryan, B., Fremantle, P. ja Obermaier, D. 2017. *The Technical Foundations of IoT*. English. Artech House.

Amazon 2020. *AWS IoT*. URL: https://aws.amazon.com/iot/. Viitattu 3.11.2020.

Anderson, J. 2020. *An Intro to Threading in Python*. URL: https://realpython.com/intro-to-python-threading/. Viitattu 3.11.2020.

Biron, J. ja Follett, J. 2016. *Foundational Elements of an IoT Solution*. English. O'Reilly Media, Inc.

Charlier, M., Goodman, E., Light, A., Lui, A. ja Rowland, C. 2015. *Designing Connected Products – UX for the Consumer Internet of Things*. O'Reilly Media, Inc.

Chart.js 2020a. URL: https://www.chartjs.org/. Viitattu 3.11.2020.

Chart.js 2020b. *Installation*. URL: https://www.chartjs.org/docs/latest/getting-started/installation.html. Viitattu 3.11.2020.

Chart.js 2020c. *Getting Started*. URL: https://www.chartjs.org/docs/latest/getting-started/. Viitattu 3.11.2020.

Chart.js 2020d. *Updating Charts*. URL: https://www.chartjs.org/docs/latest/developers/updates.html. Viitattu 3.11.2020.

Chou, T. 2016. *Precision: Principles, Practices and Solutions for the Internet of Things*. English. Cloudbook, Inc.

Cirani, S., Ferrari, G., Picone, M. ja Veltri, L. 2019. *Internet of Things - Architectures, Protocols and Standards*. English. John Wiley & Sons Ltd.

Costa, C. D. 2020. *Python vs. Node.JS: Which One Is Best For Your Project?.* URL: https://towardsdatascience.com/python-vs-node-js-which-one-is-best-for-your-project-e98f2c63f020. Viitattu 4.11.2020.

Crunchbase 2020. *Firebase*. URL: https://www.crunchbase.com/organization/firebase. Viitattu 3.11.2020.

Dexter Industries 2020. *Running Python Programs at boot on a Raspberry Pi*. URL: https://www.dexterindustries.com/howto/auto-run-python-programs-on-the-raspberry-pi/. Viitattu 3.11.2020.

Efficient IP 2020. *What is DHCP?.* URL: https://www.efficientip.com/what-is-dhcp-and-why-is-it-important/. Viitattu 3.11.2020.

Floerkemeier, C. ja Mattern, F. 2010. *From the Internet of Computers to the Internet of Things*. English. Springer-Verlag Berlin Heidelberg. URL: https://vs.inf.ethz.ch/publ/papers/Internet-of-things.pdf

Fuller, JR. 2020. *The 4 stages of an IoT architecture*. URL: https://techbeacon.com/enterprise-it/4-stages-iot-architecture. Viitattu 3.11.2020.

Google 2020a. *Cloud IoT Core*. URL: https://cloud.google.com/iot-core. Viitattu 3.11.2020.

Google 2020b. *Firebase*. URL: https://firebase.google.com/. Viitattu 3.11.2020.

Google 2020c. *Firebase Machine Learning*. URL: https://firebase.google.com/docs/ml. Viitattu 3.11.2020.

Google 2020d. *Pricing plans*. URL: https://firebase.google.com/pricing. Viitattu 3.11.2020.

Google 2020e. *Google Analytics*. URL: https://firebase.google.com/docs/analytics. Viitattu 3.11.2020.

Google 2020f. *Choose a Database*. URL: https://firebase.google.com/docs/database/rtdb-vs-firestore. Viitattu 3.11.2020.

Google 2020g. *Cloud Firestore*. URL: https://firebase.google.com/docs/firestore. Viitattu 3.11.2020.

Google 2020h. *Cloud Firestore Data model.* URL: https://firebase.google.com/docs/firestore/data-model. Viitattu 3.11.2020.

Google 2020i. *Cloud Firestore.* URL: https://firebase.google.com/docs/firestore. Viitattu 3.11.2020.

Google 2020j. *Cloud Firestore locations*. URL: https://firebase.google.com/docs/firestore/locations#default-cloud-location. Viitattu 3.11.2020.

Google 2020k. *Add the Firebase Admin SDK to your server*. URL: https://firebase.google.com/docs/admin/setup#initialize-sdk. Viitattu 3.11.2020.

Google 2020l. *Get started with Cloud Firestore*. URL: https://firebase.google.com/docs/firestore/quickstart#python. Viitattu 3.11.2020.

Google 2020m. *Add data to Cloud Firestore*. URL: https://firebase.google.com/docs/firestore/manage-data/add-data. Viitattu 3.11.2020.

Google 2020n. *Supported data types*. URL: https://firebase.google.com/docs/firestore/manage-data/data-types. Viitattu 3.11.2020.

Google 2020o. *Get data with Cloud Firestore*. URL: https://firebase.google.com/docs/firestore/query-data/get-data. Viitattu 3.11.2020.

Google 2020p. *Get realtime updates with Cloud Firestore*. URL: https://firebase.google.com/docs/firestore/query-data/listen. Viitattu 3.11.2020.

Google 2020q. *Delete data from Cloud Firestore*. URL: https://firebase.google.com/docs/firestore/manage-data/delete-data. Viitattu 3.11.2020.

Google 2020r. *Firebase Authentication*. URL: https://firebase.google.com/docs/auth. Viitattu 3.11.2020.

Google 2020s. *Firebase Security Rules*. URL: https://firebase.google.com/docs/rules. Viitattu 3.11.2020.

Google 2020t. *Structuring Cloud Firestore Security Rules*. URL: https://firebase.google.com/docs/firestore/security/rules-structure. Viitattu 3.11.2020.

Google 2020u. *Cloud Functions for Firebase*. URL: https://firebase.google.com/docs/functions. Viitattu 3.11.2020.

Google 2020v. *What can I do with Cloud Functions?*. URL: https://firebase.google.com/docs/functions/use-cases. Viitattu 3.11.2020.

Google 2020w. *APIs & server client library reference*. URL: https://cloud.google.com/firestore/docs/apis. Viitattu 3.11.2020.

Google 2020x. *Firebase CLI reference*. URL: https://firebase.google.com/docs/cli. Viitattu 3.11.2020.

Google 2020y. *Get started with Cloud Firestore*. URL: https://firebase.google.com/docs/firestore/quickstart#web. Viitattu 3.11.2020.

Google 2020z. *Get Started with Firebase Authentication on Websites*. URL: https://firebase.google.com/docs/auth/web/start. Viitattu 3.11.2020.

Google 2020å. *Google Cloud IoT solutions*. URL: https://cloud.google.com/solutions/iot. Viitattu 5.11.2020.

Google 2020ä. *Google Firebase Console*. URL: https://console.firebase.google.com. Viitattu 5.11.2020.

Gour, R. *Top 10 Applications of IoT*. URL: https://dzone.com/articles/top-10-uses-of-the-internet-of-things. Viitattu 3.11.2020.

Hanson, J. 2020. *What is http Long Polling?.* URL: https://www.pubnub.com/blog/http-long-polling/. Viitattu 3.11.2020.

i-SCOOP 2020a. *IoT gateways*. URL: https://www.i-scoop.eu/internet-of-things-guide/iot-gateways/. Viitattu 3.11.2020.

i-SCOOP 2020b. *IoT platforms*. URL: https://www.i-scoop.eu/internet-of-things-guide/iot-platform-market-2017-2025/. Viitattu 3.11.2020.

Javascript.info 2020. *An Introduction to JavaScript*. URL: https://javascript.info/intro. Viitattu 3.11.2020.

KaaIoT 2020. *What is an IoT platform?*. URL: https://www.kaaproject.org/blog/what-is-iot-platform. Viitattu 3.11.2020.

Leverege 2020. *Introduction to UIs & UX for IoT*. URL: https://www.leverege.com/iot-ebook/ui-and-ux-design-iot. Viitattu 3.11.2020.

Linna, J. 2020. *Visualization Provides Better Understanding of IoT Data*. URL: https://www.solita.fi/en/blogs/visualisation-provides-better-understanding-of-iot-data/. Viitattu 3.11.2020.

Microsoft 2020a. *How does Websocket work*. URL: https://docs.microsoft.com/en-us/azure/application-gateway/application-gateway-websocket. Viitattu 3.11.2020.

Microsoft 2020b. *Azure IoT*. URL: https://azure.microsoft.com/en-us/overview/iot/. Viitattu 3.11.2020.

MongoDB 2020a. *What is NoSQL?*. URL: https://www.mongodb.com/nosql-explained. Viitattu 3.11.2020.

MongoDB 2020b. *Introduction to MongoDB*. URL: https://docs.mongodb.com/manual/introduction/. Viitattu 3.11.2020.

MongoDB 2020c. *The Internet of Things*. URL: https://www.mongodb.com/use-cases/internet-of-things. Viitattu 3.11.2020.

MongoDB 2020d. *The Internet of Things*. URL: https://www.mongodb.com/use-cases/internet-of-things. Viitattu 3.11.2020.

Mozilla 2020a. *HTML basics. URL*: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics. Viitattu 3.11.2020.

Mozilla 2020b. *What is DOM?.* URL: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction. Viitattu 3.11.2020.

OAS 2020. *What is an IoT Gateway?.* URL: https://openautomationsoftware.com/open-automation-systems-blog/what-is-an-iot-gateway/. Viitattu 3.11.2020.

OmniSci 2020. *Interoperability*. URL: https://www.omnisci.com/technical-glossary/interoperability. Viitattu 3.11.2020.

Openscource.com 2020. *What is a Raspberry Pi?.* URL: https://opensource.com/resources/raspberry-pi. Viitattu 3.11.2020.

Pallets Projects 2020. *Flask*. URL: https://palletsprojects.com/p/flask/. Viitattu 5.11.2020.

Philips 2020a. *How Philips Hue works*. URL: https://www.philips-hue.com/en-us/how-it-works. Viitattu 3.11.2020.

Philips 2020b. *What Hue developers can do?.* URL: https://developers.meethue.com/explore/why-develop-for-hue/. Viitattu 3.11.2020.

Philips 2020c. *Core Concepts*. URL: https://developers.meethue.com/develop/get-started-2/core-concepts/. Viitattu 3.11.2020.

Philips 2020d. *Get started*. URL: https://developers.meethue.com/develop/get-started-2/. Viitattu 3.11.2020.

Poyen, F. 2019. *Annual Technical Volume of Computer Engineering Division Board: Theme: Role of IoT in Make in India.* English. The Institution of Engineers (India). ss. 42-47. URL: https://www.researchgate.net/publication/330200556_Raspberry_Pi_and_its_Use_in_IoT_Applications. Viitattu 3.11.2020.

Pozdnyakov, M. ja Shalamov, A. 2020. *Sensors For The Web!.* URL: https://developers.google.com/web/updates/2017/09/sensors-for-the-web. Viitattu 7.11.2020.

PyPA 2020. *Installing Packages*. URL: https://packaging.python.org/tutorials/installing-packages/. Viitattu 3.11.2020.

Python 2020. *JSON encoder and decoder*. URL: https://docs.python.org/3/library/json.html. Viitattu 3.11.2020.

Rasbian 2020. *Welcome to Raspbian*. URL: https://www.raspbian.org/FrontPage. Viitattu 3.11.2020.

Raspberry Pi 2020a. *Raspberry Pi 3 Model B*. URL: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. Viitattu 3.11.2020.

Raspberry Pi 2020b. *Raspberry Pi OS*. URL: https://www.raspberrypi.org/downloads/raspberry-pi-os/. Viitattu 3.11.2020.

Requests 2020. *Requests: HTTP for Humans*. URL: https://requests.readthedocs.io/en/master/. Viitattu 3.11.2020.

Singh, V. 2020. *What is frameworks?.* URL: https://hackr.io/blog/what-is-frameworks. Viitattu 3.11.2020.

Techopedia 2020. *Datagram*. URL: https://www.techopedia.com/definition/6766/datagram. Viitattu 3.11.2020.

Tensorflow 2020. URL: https://www.tensorflow.org/. Viitattu 5.11.2020.

ThingsBoard 2020a. URL: https://thingsboard.io/. Viitattu 3.11.2020.

ThingsBoard 2020b. *Smart farming dashboard.* URL: https://thingsboard.io/smart-farming/. Viitattu 3.11.2020.

Vue CLI 2020. *Installation*. URL: https://cli.vuejs.org/guide/installation.html. Viitattu 3.11.2020.

Vue Router 2020. *Getting Started*. URL: https://router.vuejs.org/guide/. Viitattu 3.11.2020.

Vue.js 2020a. *Introduction.* URL: https://vuejs.org/v2/guide/. Viitattu 3.11.2020.

Vue.js 2020b. *Template Syntax*. URL: https://vuejs.org/v2/guide/syntax.html. Viitattu 3.11.2020.

Vue.js 2020c. *The Vue Instance*. URL: https://vuejs.org/v2/guide/instance.html. Viitattu 3.11.2020.

W3Schools 2020. *ECMAScript 2015*. URL: https://www.w3schools.com/js/js_es6.asp. Viitattu 3.11.2020.